Machine learning and combinatorial optimization algorithms, with applications to railway planning

Guillaume Dalle

École des Ponts ParisTech

THÈSE DE DOCTORAT
de l'École des Ponts ParisTech
Mathématiques appliquées

Machine learning and combinatorial optimization algorithms, with applications to railway planning

École doctorale N°532, Mathématiques et STIC

Thèse préparée au laboratoire CERMICS
École des Ponts ParisTech, Batiment Coriolis, 6 et 8 avenue Blaise Pascal
Cité Descartes, Champs-sur-Marne, 77455 Marne-la-Vallée Cedex 2

Thèse soutenue le 16 décembre 2022 par
Guillaume DALLE

Composition du jury:

Élisabeth GASSIAT Professeure, Université Paris-Saclay	*Présidente du jury*
Pierre ALQUIER Research scientist, RIKEN AIP	*Rapporteur*
Éric MOULINES Professeur, École polytechnique	*Rapporteur*
Mathieu BLONDEL Senior research scientist, Google Research	*Examinateur*
Jérôme MALICK Directeur de recherche, Université Grenoble Alpes	*Examinateur*
Frédéric MEUNIER Professeur, École des Ponts ParisTech	*Directeur de thèse*
Yohann DE CASTRO Professeur, École Centrale de Lyon	*Co-directeur de thèse*
Axel PARMENTIER Chercheur, École des Ponts ParisTech	*Co-encadrant de thèse*

À Henri et Gérard, parce qu'ils seraient si fiers de moi.

À Val, parce que je suis si fier de toi.

Remerciements

I'm Chandler. I make jokes when I'm uncomfortable.

Chandler Bing
Friends – S6E24
The One with the Proposal, Part I (2000)

Avec tout le sérieux[1] dont je suis capable, j'adresse mes premiers remerciements à deux encadrants de haute volée. Axel et Yohann, chacun à votre manière, vous m'avez insufflé le goût de la recherche, des beaux résultats et des preuves infâmes, des raisonnements rigoureux et des approximations barbares. Vous avez surtout fait preuve d'écoute et d'humanité dans un contexte sanitaire et personnel très compliqué pour moi. J'ai eu de la chance de vous avoir comme mentors... mais forcément j'ai attrapé certaines de vos manies. Désormais, comme Yohann, je ferme les yeux quand j'essaie de dire un truc malin. Et comme Axel, je gribouille rageusement au stylo rouge le moindre papier qu'on me donne à relire. Espérons que ce soit temporaire.

Merci à Pierre Alquier et Éric Moulines d'avoir accepté de relire le présent manuscrit. Merci également à Élisabeth Gassiat qui a présidé mon jury de soutenance, aux côtés de Mathieu Blondel et Jérôme Malick. Vos retours sur mes résultats ont été précieux, et nos discussions lors de la soutenance m'ont grandement intéressé. Je suis convaincu que l'Histoire oubliera les problèmes de WiFi, pour ne retenir que l'excellent buffet (et les blagues sur les trains).

Quand je dis aux gens que je suis chercheur en mathématiques, la réaction typique est un mélange d'admiration et de dégoût. Pourtant ce n'est pas si surprenant : depuis mon plus jeune âge, je prends plaisir à apprendre des trucs, puis à baratiner les autres en leur déballant ce que j'ai appris. Ça en dit sans doute long sur moi et mon environnement familial, mais aussi sur la chouette cohorte d'enseignant·es dont j'ai bénéficié tout au long de mon parcours. Merci donc à Mesdames Cariou, Brun, Aubelle, Velay et Dunlop, et à Messieurs Petitcolin, Troesch et Tosel. Sans vous, les sciences auraient été bien moins fun.

Une thèse, ce n'est pas qu'un sujet, c'est aussi un endroit. Or nous sommes en 2023 après Jésus-Christ, et toute l'intelligentsia parisienne est regroupée à Saclay. Toute ? Non ! Un petit village, peuplé d'irréductibles matheux et matheuses, persiste encore et toujours à prendre le RER A : bienvenue au CERMICS.

Dans le rôle d'Abraracourcix, j'ai nommé Isabelle. Affrontant sans relâche les difficultés administratives, elle veille quotidiennement sur un cadre de travail presque aussi paisible que la campagne armoricaine. Mais que serait le chef de village sans son fidèle druide ? Et ça tombe bien, Stéphanie fait un excellent Panoramix. Personne ne connaît mieux qu'elles les secrets oubliés des notes de frais, les arcanes mystérieuses des départs en mission, le langage obscur des codes de la bibliothèque.

Pour remettre l'oeuvre de Goscinny et Uderzo au goût du jour, le doyen Agecanonix est incarné collectivement par les titulaires du labo. Ils et elles sont dépositaires d'une sagesse millénaire, et chaque interaction permet de s'en abreuver. Je pense notamment à Frédéric, qui m'a donné ma première opportunité d'enseigner, ou à Antoine, avec qui j'ai fréquemment parlé Julia. Les conseils de Tony, Eric et Vincent m'ont aussi beaucoup aidé lorsque je cherchais ma voie. Je salue au passage les autres chercheurs avec qui j'ai collaboré, principalement Thibaut Vidal et Mohamed Tarek.

[1] Limité

Ordralphabétix et Cétautomatix, le poissonnier et le forgeron, sont des allégories du conflit éternel entre les étudiant·es du deuxième et du troisième étage. Heureusement, les membres de chaque faction ne sont pas rancunier·es, et se retrouvent joyeusement pour le banquet quotidien à la cantine (depuis qu'elle a rouvert). L'ambiance chaleureuse qui donne envie d'aller au bureau, c'est à vous qu'on la doit. Je tiens tout particulièrement à rendre hommage à Louis et Léo, mes camarades de galère face aux camions récalcitrants et aux bugs de différentiation automatique. Espérons que le CIel ne vous tombe pas sur la tête !

Tel Obélix (dont je partage le palais raffiné et le sens du style), je m'aventurais parfois au-delà des frontières de mon village. Lors de mes escarmouches avec l'administration romaine, j'ai trouvé en la personne d'Élodie un parfait Idéfix, contrepoids subtil de mon approche un peu bourrue. Notre mandat de délégué·es des doctorant·es fut l'occasion de contribuer à la bonne marche de l'établissement, et d'en découvrir les rouages. On a beau dire, les légions de César qui en ont la charge font du bon boulot, et je leur en suis reconnaissant.

Régulièrement, mon chemin m'a mené jusqu'à Lutèce pour y parler menhirs avec des collègues du métier. La SNCF (Sculpture Néolithique de Cailloux Fuselés) m'a fourni l'inspiration dont j'avais besoin, grâce à des échanges très riches que j'espère approfondir à l'avenir. Ma gratitude va donc à Bertrand Houzel, Clément Raoux et Florian Chassagne côté Réseau, Philippe De Laharpe, Benjamin Huteau et Claire Messner côté Voyageurs, ainsi qu'à François Ramond, David De Almeida et à l'ensemble du cluster ORE. J'ai hâte de continuer à travailler avec vous !

En attendant, j'ai eu la chance de faire un tour en Amérique, puis de poser mes valises en Helvétie. Lors de mon retour en Gaulle, c'est avec joie que je retrouverai l'École pour rejoindre le LVMT. Merci à Sophie Mougard et Pierre Zembri de m'avoir accordé leur confiance.

Vous le savez sûrement : en 2019 quelqu'un a mangé du sanglier pas frais, et ensuite on a passé un petit bout de temps à la maison. Afin d'atténuer la solitude des doctorant·es durant cette période, un serveur Discord baptisé *PhD Students* a vu le jour. Dans les multiples recoins de cet endroit merveilleux, au détour de visios studieuses ou de soirées jeux, j'ai gagné tout un tas d'ami·es que je n'attendais pas. Petit coucou à Eva, qui m'a aidé à assumer ma vocation de barde, et à Mathieu, que je harcèle sans remords avec des questions désagréables sur ses packages Julia.

En parlant de Julia, voilà un autre moyen de faire des rencontres sans montrer son attestation Covid aux soldats romains. C'est une belle communauté, accueillante et inclusive, dont le seul défaut serait sans doute... un prosélytisme excessif ? Je remercie tout spécialement Alan Edelman de m'avoir invité au MIT, l'organisation JuliaGraphs, avec laquelle j'ai fait mes premiers pas dans l'*open source*, ainsi que le comité d'organisation des JuliaCon, qui effectue chaque année un travail exceptionnel.

Vous l'avez deviné, malgré ce village haut en couleurs, je ne serais rien sans mon Astérix. Et ce compagnon de tous les jours, c'est l'énigmatique Godalle Marmanthier. Mathématicien émérite, ami fidèle, fonctionnaire dévoué, memeur infatigable, grand amateur de jeux de société et de plats à base de fromage, cet étrange groupuscule (un genre de Bourbaki des temps modernes) joue un rôle central dans ma vie depuis moult éons. Même si Slack efface périodiquement sa mémoire, sa légende survivra ! Clément, Éloïse, Maxime et Pierre, je vous lève haut mon verre (de jus de fruit). Mention honorable pour Alexis, Induja, Florentin et Anne, dont les machinations éhontées sur *Among Us* ont égayé bien des soirées de confinement.

Les bandes dessinées restent assez floues sur la famille d'Obélix. La mienne en revanche m'a apporté un soutien indéniable pendant ces longues années d'études. Frère, parents, grand-parents,

oncles et tantes, cousins et cousines ont été tour à tour des exemples, des refuges, des supporters, des confidents. Là encore, je me sais très chanceux d'avoir été si bien entouré.

Enfin, j'ai une pensée affectueuse pour Marine, alias Falbala, avec qui j'ai partagé toute ma vie d'adulte. La thèse fut une période difficile, mais j'ai toujours pu compter sur toi. Espérons que l'avenir soit plus clément pour nous deux.

Abstract (français)

> Ah ! non ! c'est un peu court, jeune homme ! On pouvait dire... Oh ! Dieu !... bien des choses en somme...
>
> *Cyrano*
> Cyrano de Bergerac (1897)

Cette thèse en mathématiques appliquées mélange l'apprentissage statistique et l'optimisation combinatoire. Elle associe des avancées théoriques à des algorithmes efficaces, introduisant au passage plusieurs librairies *open source* en Julia. Grâce à une collaboration avec la SNCF, trois applications au transport ferroviaire sont présentées : prédiction des pannes, propagation des retards et allocation des voies.

La Partie I décrit les fondements mathématiques et l'implémentation de plusieurs ingrédients utiles par la suite : différenciation implicite, processus ponctuels, modèles de Markov cachés, recherche d'itinéraires multi-agent. Notre code en libre accès remplit un vide dans l'écosystème Julia, combinant facilité d'usage et haute performance.

La Partie II contient des contributions théoriques liées aux statistiques et à la prise de décision. Notre étude d'un processus autorégressif vectoriel partiellement observé met en évidence des bornes supérieure et inférieure cohérentes sur l'erreur d'estimation. Une exploration des couches d'optimisation combinatoire pour l'apprentissage profond nous permet de développer le package `InferOpt.jl`, qui unifie et approfondit l'état de l'art. Pour étendre ces méthodes à des couches d'optimisation multi-objectif, nous construisons une nouvelle théorie de l'optimisation lexicographique convexe.

La Partie III s'inspire des deux précédentes pour traiter des problèmes ferroviaires concrets. Nous proposons un modèle hiérarchique pour les pannes de trains, une approche graphique pour la propagation des retards, et de nouvelles perspectives pour l'allocation des voies, avec le challenge Flatland comme terrain d'expérimentation.

Mots-clefs : apprentissage statistique, optimisation combinatoire, modèles à variables latentes, couches différentiables d'optimisation, langage Julia, planification ferroviaire

Abstract

Let me explain... no, there is too much.
Let me sum up.

Inigo Montoya
The Princess Bride (1987)

This thesis investigates the frontier between machine learning and combinatorial optimization, two active areas of applied mathematics research. We combine theoretical insights with efficient algorithms, and develop several open source Julia libraries. Inspired by a collaboration with the *Société nationale des chemins de fer français* (SNCF), we study high-impact use cases from the railway world: train failure prediction, delay propagation, and track allocation.

In Part I, we provide mathematical background and describe software implementations for various tools that will be needed later on: implicit differentiation, temporal point processes, Hidden Markov Models and Multi-Agent Path Finding. Our publicly-available code fills a void in the Julia package ecosystem, aiming at ease of use without compromising on performance.

In Part II, we highlight theoretical contributions related to both statistics and decision-making. We consider a Vector AutoRegressive process with partial observations, and prove matching upper and lower bounds on the estimation error. We unify and extend the state of the art for combinatorial optimization layers in deep learning, gathering various approaches in a Julia library called `InferOpt.jl`. We also seek to differentiate through multiobjective optimization layers, which leads to a novel theory of lexicographic convex analysis.

In Part III, these mathematical and algorithmic foundations come together to tackle railway problems. We design a hierarchical model of train failures, propose a graph-based framework for delay propagation, and suggest new avenues for track allocation, with the Flatland challenge as a testing ground.

Keywords: machine learning, combinatorial optimization, latent variable models, differentiable optimization layers, Julia language, railway planning

Contents

1

Introduction (français)

D'accord, faisons comme ça !

Hubert Bonisseur de La Bath
OSS 117 : Rio ne répond plus (2009)

Contents

Cette thèse en mathématiques appliquées mélange l'apprentissage statistique et l'optimisation combinatoire. Elle associe des avancées théoriques à des algorithmes efficaces, introduisant au passage plusieurs librairies *open source* en Julia. Grâce à une collaboration avec la SNCF, plusieurs applications au transport ferroviaire sont présentées : prédiction des pannes, propagation des retards et allocation des voies. Cependant, notre travail peut être transposé à de nombreux autres contextes industriels.

Dans le présent chapitre, nous motivons notre recherche par des problèmes de planification ferroviaire, avant d'énumérer nos principales contributions mathématiques et informatiques.

1.1 Contexte industriel : une vue d'ensemble des opérations ferroviaires

1.1.1 Pourquoi étudier les chemins de fer ?

Sécheresses. Canicules. Feux de forêt. Tempêtes. Inondations. Alors que le changement climatique s'accélère, réduire les émissions de gaz à effet de serre doit devenir une priorité absolue, en particulier dans les pays industrialisés. En France, l'empreinte carbone d'un individu moyen s'élève à 9 tonnes d'équivalent CO_2 par an, alors que l'objectif de neutralité pour 2050 est fixé à 2 tonnes[1]. Les changements nécessaires pour atteindre cet objectif vont transformer tous les aspects de notre vie quotidienne.

Actuellement, le secteur des transports génère à lui seul environ un tiers des émissions, essentiellement à cause du transport routier et aérien. Grâce à leur grande capacité et à leur efficacité énergétique (sans oublier une production électrique très décarbonée), les trains français rejettent beaucoup moins de CO_2 par kilomètre et par passager que les voitures ou les avions. Augmenter la part modale du rail semble donc un objectif louable, mais comment inciter davantage de personnes à prendre le train ? Bien sûr, une partie de la réponse réside dans les politiques publiques et les décisions d'investissement, mais l'attrait des chemins de fer n'est pas seulement une question d'argent et d'infrastructures. La qualité du service dépend également de la rapidité et de la résilience des décisions, et c'est là que les mathématiques appliquées ont un rôle à jouer.

1.1.2 Processus de planification et ressources

Même si les voyageurs ne s'en rendent pas toujours compte, déplacer un train d'un point A à un point B est loin d'être facile. En coulisses, un processus de planification complexe se déroule, comme le décrit Schlechte (2012, Chapitre I). Ce processus fait intervenir deux acteurs principaux : les entreprises ferroviaires (comme SNCF *Voyageurs*), chargées de faire rouler les trains, et les gestionnaires d'infrastructure (comme SNCF *Réseau*), chargés de coordonner l'accès au réseau. Ces acteurs s'efforcent d'organiser chaque voyage en rassemblant un certain nombre de ressources, que nous décrivons à présent.

L'*infrastructure* est une ressource centrale pour le transport ferroviaire. Contrairement aux voitures et aux avions, les trains se déplacent sur un réseau très contraint de voies et de gares. Leurs distances de freinage étant importantes, ils doivent maintenir en permanence un grand intervalle de sécurité pour éviter les collisions. Dès que l'on dépasse de faibles vitesses, cet intervalle de sécurité s'étend au-delà du champ de vision humain. Par conséquent, chaque trajet doit réserver à l'avance un itinéraire (une séquence de sections de voie), dont on doit s'assurer qu'il est exempt de conflits. Il serait inutile de monopoliser un itinéraire complet pour toute la durée du voyage, c'est pourquoi les sections de voie individuelles sont attribuées puis libérées au fur et à mesure du passage des trains. Cela permet de respecter les intervalles de sécurité, tout en assurant un débit élevé.

Le *matériel roulant* et le *personnel* sont d'autres ressources essentielles. Un trajet donné ne peut avoir lieu qu'avec un véhicule répondant à des exigences spécifiques (capacité, nombre de voitures, type de moteur), doté d'un personnel adéquat (conducteur/conductrice, contrôleur/contrôleuse, vendeur/vendeuse de nourriture). Enfin, et surtout, les passagers/passagères peuvent également être considérés comme une ressource, car un train est inutile (et coûteux) s'il voyage à vide. Les

[1] https://www.statistiques.developpement-durable.gouv.fr/edition-numerique/chiffres-cles-du-climat-2022/

départs sont souvent retardés pour rendre les correspondances possibles, tout comme ils peuvent être retardés pour attendre un véhicule ou un membre d'équipage.

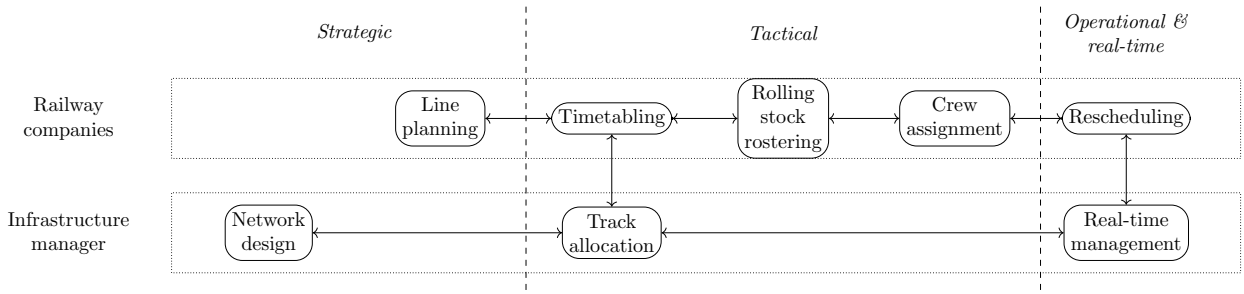


Figure 1.1: Principales phases de la planification ferroviaire
Adapté d'après Schlechte (2012) avec permission de l'auteur

La planification ferroviaire s'étend à travers plusieurs échelles de temps. Au niveau *stratégique* (plusieurs années à l'avance), les décisions structurelles telles que la conception du réseau et la planification des lignes sont prises. Ensuite, au niveau *tactique* (plusieurs mois à l'avance), la construction des horaires et l'allocation des voies se font en parallèle. Il s'agit d'une condition nécessaire à la constitution des rotations de matériel roulant et à l'affectation des équipes. Ensuite, au niveau *opérationnel* (plusieurs jours à l'avance), des ajustements de dernière minute sont apportés à la grille horaire. Enfin, les perturbations du trafic en temps réel sont surveillées et traitées par des centres de contrôle spécialisés. Ces phases sont résumées sur la Figure 1.1.

1.1.3 Optimisation et apprentissage pour un système résilient

À chaque étape de la planification, les entreprises ferroviaires et les gestionnaires d'infrastructure doivent répondre à une question difficile : quelle est la meilleure utilisation possible de nos ressources ? Cette décision implique de nombreuses variables à définir, plusieurs contraintes à satisfaire, ainsi qu'une fonction de coût à minimiser, c'est pourquoi nous l'appelons un *problème d'optimisation*. Pour ne rien arranger, ce problème est *combinatoire* : il possède un ensemble fini mais exponentiellement grand de solutions possibles, de sorte que l'énumération exhaustive devient impossible en temps raisonnable. Le domaine de l'optimisation combinatoire (CO) est consacré à l'étude de tels problèmes, et il a donné lieu à de nombreux algorithmes efficaces depuis les années 1950 (Korte and Vygen 2006). Les applications de la CO aux chemins de fer sont presque aussi anciennes que le domaine lui-même, et restent encore très pertinentes aujourd'hui (Borndörfer et al. 2018).

Cependant, les algorithmes de CO ne sont pas toujours suffisants lorsqu'ils sont utilisés tels quels. Tout d'abord, la formulation usuelle des problèmes suppose que l'avenir ne réserve aucune surprise, et que tout se déroule comme prévu. Pourtant, l'expérience montre que des incidents se produisent, et nous devons donc les anticiper en amont. Deuxièmement, la plupart des algorithmes d'optimisation sont conçus pour fonctionner sur une grande variété d'instances. Mais en réalité, nous nous concentrons sur un cas d'usage bien précis (par exemple, un réseau ferroviaire spécifique). Nous aimerions donc que notre solveur y soit aussi performant que possible, même si cela implique de renoncer à la généralité. L'apprentissage automatique (ML) est la clé pour surmonter ces deux obstacles. Il englobe une vaste famille de méthodes qui permettent d'extraire des informations à partir de données brutes. Et heureusement, les chemins de fer génèrent beaucoup de données : des

heures de départ et d'arrivée à la surveillance de l'état des trains ou au comptage des passagers, les mesures ne manquent pas. Le véritable défi consiste à leur donner du sens, avant de les exploiter au sein d'algorithmes d'optimisation.

1.1.4 Trois problèmes ferroviaires importants

Dans cette thèse, notre objectif final est d'effectuer l'allocation des voies. Nous voulons prendre une grille horaire en entrée, et affecter à chaque train un itinéraire réalisable à travers le réseau. Pour générer des réseaux et des horaires réalistes, nous nous appuyons sur le simulateur du challenge Flatland (Mohanty et al. 2020), dont un exemple est présenté sur la Figure 1.2. L'allocation des voies pour Flatland est un cas particulier de recherche d'itinéraires multi-agent (MAPF). Dans ce problème classique, un ensemble d'agents doit trouver des chemins mutuellement compatibles à travers un graphe. Le but de chacun est d'atteindre sa destination le plus rapidement possible, le tout sans causer de collisions.

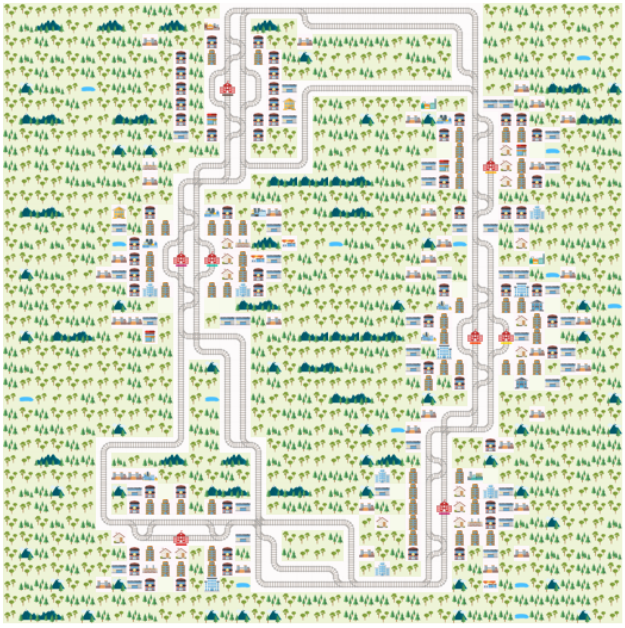


Figure 1.2: Exemple de carte générée par le simulateur Flatland (voir Chapitre 11)

Le Chapitre 5 étudie cette question de manière plus détaillée. Comme nous le verrons, il s'agit d'un problème **NP**-difficile, dont les principales méthodes de résolution reposent sur des techniques de décomposition, des approximations polynomiales ou une recherche locale. Nous adaptons et implémentons un algorithme de descente à grands voisinages, qui s'exécute rapidement sur de très grandes instances, et nous le testons sur un jeu de *benchmark*. Ensuite, dans le Chapitre 11, nous décrivons une façon d'améliorer cet algorithme en apprenant à partir de données simulées par Flatland.

Les Chapitres 5 et 11 se concentrent sur la version déterministe de l'allocation des voies. Que se passe-t-il si nous devons également faire face à des retards imprévisibles et stochastiques ? Nous aurons alors besoin de modèles statistiques, pour fournir des prédictions fiables à nos algorithmes d'optimisation.

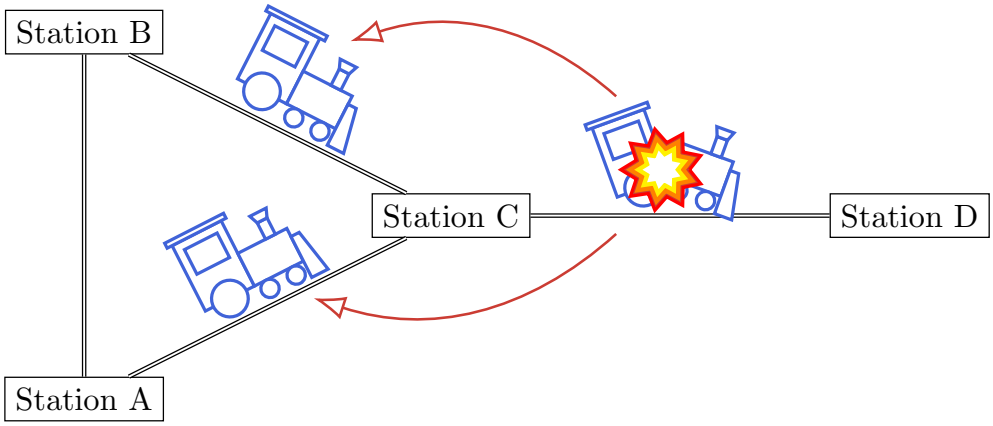


Figure 1.3: Deux types de retards : primaires (explosion $\implies$ panne) et secondaires (flèche $\implies$ propagation)

Comme l'illustre la Figure 1.3, les retards de trains se répartissent en deux catégories : primaires et secondaires. Les retards primaires sont exogènes : ils sont causés par des incidents tels que des dysfonctionnements mécaniques, des intempéries ou des comportements humains inattendus. À l'inverse, les retards secondaires sont endogènes : ils sont générés par les interactions entre les trains sur le réseau. Pour parvenir à une allocation résiliente des voies, nous devons apprendre à prévoir ces deux types de retards.

Les défaillances du matériel roulant sont parmi les principales causes des retards primaires, c'est pourquoi nous étudions la prédiction des pannes dans le Chapitre 9. Sur la base des données de surveillance d'une flotte de trains régionaux, nous essayons d'anticiper leur dégradation et de prédire leur prochaine intervention corrective au centre de maintenance. Dans des travaux futurs, ces informations nous aideront à identifier les véhicules les plus vulnérables, et à les affecter aux itinéraires les plus sûrs.

L'effet papillon donnant lieu à des retards secondaires est appelé *propagation des retards*, et nous l'étudions dans le Chapitre 10. Pour quantifier ce phénomène, notre méthode ne repose pas sur des données industrielles confidentielles, mais uniquement sur les heures de départ et d'arrivée. Nous l'appliquons aux relevés horaires d'un système de tramway suisse, et nous retrouvons des intuitions cohérentes sur la dynamique de la congestion des voies. Dans des travaux futurs, ces informations nous aideront à identifier les zones les plus critiques du réseau, afin d'éviter de les surcharger.

1.2 Contributions scientifiques : prise de décision rapide basée sur les données

Avant de nous plonger dans les applications ferroviaires, nous utilisons les Chapitres 3 à 8 pour développer des outils théoriques et algorithmiques de prise de décision *data-driven*. Dans l'optique de combiner ML et CO, nous devons répondre aux questions suivantes :

1. Comment extraire des informations pertinentes de données bruitées en grande dimension ?
2. Comment utiliser ces informations pour améliorer les algorithmes d'optimisation ?

Nous énumérons maintenant nos principales contributions à chacun de ces sujets de recherche.

1.2.1 Apprentissage en grande dimension avec des variables cachées

Le premier thème récurrent dans nos travaux est la notion de variable cachée. Elle apparaît dans la prédiction des pannes (Chapitre 9) pour modéliser l'état de dégradation d'un train, puis à nouveau dans la propagation des retards (Chapitre 10) pour capturer la notion de congestion du réseau. Dans ces deux contextes, nous travaillons avec des modèles temporels de forme générique

$$X_t = f_t(X_{t-1}) \qquad \text{et} \qquad Y_t = g_t(X_t) \tag{1.1}$$

Ici, X_t est le processus latent, auquel nous n'avons pas accès, tandis que Y_t est le processus observé. Les fonctions f_t et g_t sont aléatoires, appelées respectivement distributions de transition et d'émission. Dans certains cas, trouver une forme paramétrique appropriée pour f_t et g_t est loin d'être évident. Dans d'autres cas, le modèle est assez simple, mais son estimation soulève des questions statistiques non triviales.

1.2.1.1 Modélisation de flux de données complexes

Dans le contexte de la prédiction des pannes (Chapitre 9), notre principal défi est de construire un modèle de dégradation adéquat à partir des données qui nous sont fournies. Les journaux d'activité des trains sont mis à disposition quotidiennement : nous les considérons comme des variables de contrôle, qui influencent les distributions de transition et d'émission. À l'inverse, nos observations sont un flux d'événements enregistrés en temps continu, ce qui n'est pas compatible avec une formulation en temps discret. De plus, les événements et les contrôles sont représentés par des vecteurs de grande dimension, dont certaines parties peuvent être inutiles ou redondantes.

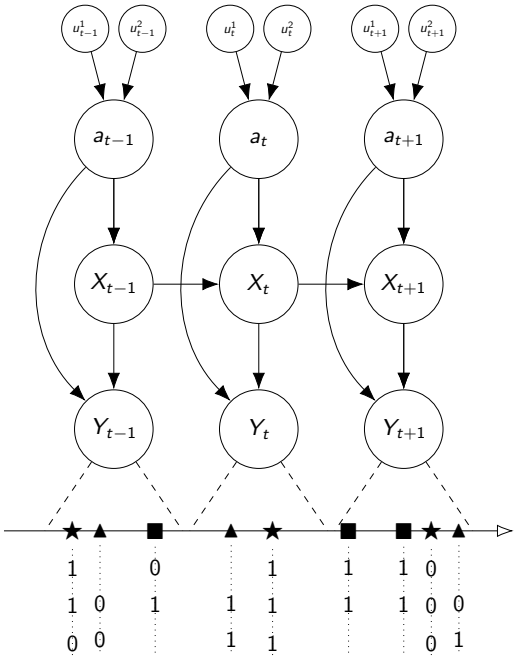
Pour surmonter ces obstacles, nous nous appuyons sur les concepts introduits par le Chapitre 4 pour définir un *modèle de Markov caché contrôlé* avec des observations de type *processus de Poisson marqué*. Ce modèle est illustré sur la Figure 1.4a : il fusionne les données en temps discret et en temps continu, sans avoir besoin de compresser l'information. Les distributions de transition et d'émission sont spécifiquement conçues pour refléter notre connaissance du processus de détérioration des trains. Des hypothèses naturelles sur les dépendances stochastiques permettent de réduire le nombre de paramètres, atténuant ainsi la malédiction de la dimension.

1.2.1.2 Obtention de garanties statistiques grâce à la parcimonie

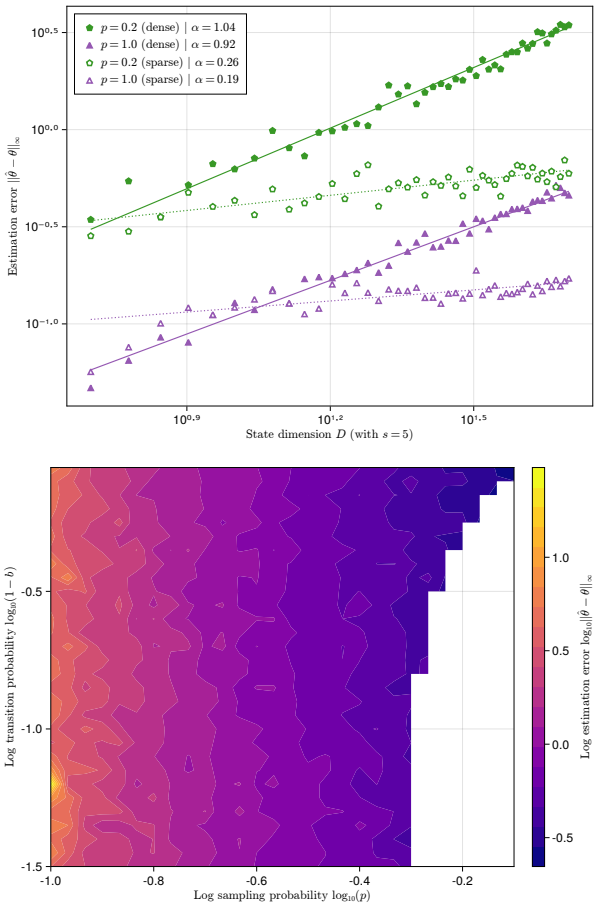
Dans le contexte de la propagation des retards (Chapitre 10), nous introduisons un nouveau modèle centré sur une variable de congestion X_t. Cette congestion se déplace dans le réseau, et influence les temps de départ et d'arrivée Y_t car elle rend les sections de voie plus ou moins difficiles à traverser. Pour modéliser X_t (le processus latent) et Y_t (le processus observé), nous utilisons une *autorégression vectorielle partiellement observée* :

$$X_t = \theta X_{t-1} + \varepsilon_t \qquad \text{et} \qquad Y_t = \Pi_t X_t + \eta_t \tag{1.2}$$

Dans l'Équation (1.2), ε_t et η_t sont des bruits blancs Gaussiens de variances respectives σ^2 et ω^2. La matrice Π_t est un masque aléatoire qui cache chaque composante de Y_t avec une probabilité $1-p$, tout en présentant d'éventuelles corrélations temporelles. Enfin, θ est une matrice de poids parcimonieuse, qui possède au maximum s coefficients non nuls par ligne. L'estimation de θ est essentielle pour comprendre la dynamique des retards, c'est pourquoi nous devons contrôler l'erreur d'estimation.



(a) HMM contrôlé + processus de Poisson = modèle hiérarchique pour la prédiction des pannes (voir Chapitre 9)



(b) L'estimation d'une autorégression vectorielle partiellement observée est plus facile avec un petit s et un grand p (voir Chapitre 6)

Figure 1.4: Apprentissage en grande dimension avec des variables cachées

Le Chapitre 6 fournit une analyse complète de ce modèle à espace d'états assez inhabituel. En prouvant une nouvelle borne supérieure (Theorem 6.4.1) et la borne inférieure minimax associée (Theorem 6.4.2), nous quantifions formellement l'erreur de l'estimateur optimal sur une séquence de longueur T :

$$\|\widehat{\theta} - \theta\|_\infty \propto \left(1 + \frac{\omega^2}{\sigma^2}\right) \frac{s}{p\sqrt{T}}$$

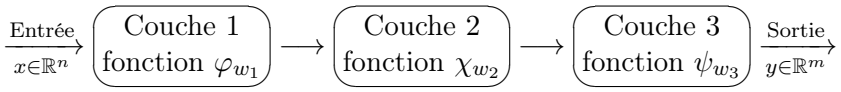
Sans surprise, la parcimonie du modèle (petit s) facilite l'estimation, tandis que la parcimonie des observations (petit p) rend l'estimation plus difficile. C'est ce que montre la Figure 1.4b.

1.2.2 Algorithmes d'optimisation dans les pipelines d'apprentissage

Le deuxième thème récurrent dans notre travail est le concept de pipeline d'apprentissage différentiable.

1.2.3 Qu'est-ce qu'un pipeline ?

Ces dernières années, l'apprentissage profond a permis d'obtenir des résultats impressionnants pour le traitement du langage naturel, la génération d'images et bien d'autres tâches autrefois considérées comme hors de portée des machines (Goodfellow, Bengio, and Courville 2016). Les réseaux de neurones fonctionnent en appliquant une succession d'opérations de base, telles que des combinaisons linéaires ou des fonctions d'activation non linéaires. Ces opérations sont appelées *couches*. Ensemble, elles forment un graphe de calcul (acyclique dirigé), aussi appelé *pipeline* :



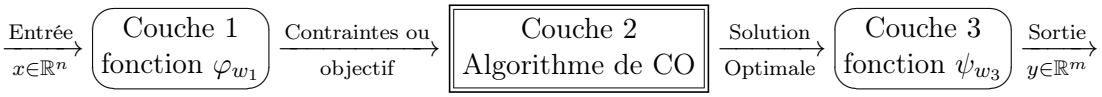
Chacune des couches possède des paramètres internes, ou *poids*, qui doivent être calibrés durant la phase d'entraînement. La plupart du temps, cet entraînement implique de minimiser une fonction de perte $\mathcal{L}_w$ définie sur N échantillons (entrée, sortie) :

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_w(x^{(i)}, \overline{y}^{(i)}).$$

La fonction $\mathcal{L}_w$ mesure une distance entre la sortie du réseau appliqué à l'entrée x et la sortie cible $\overline{y}$. Elle est généralement minimisée par descente de gradient, mais le calcul manuel des dérivées devient très fastidieux pour les pipelines complexes. C'est là que la différenciation automatique (AD) entre en scène : celle-ci calcule des dérivées numériquement exactes pour chaque couche, sans avoir besoin d'une formule analytique. Ensuite, l'AD combine ces dérivées (voir Chapitre 3) pour rétro-propager les gradients de la fonction de perte vers les poids des différentes couches.

1.2.3.1 Couches différentiables d'optimisation combinatoire

La plupart des couches de ML sont des fonctions simples et explicites, faciles à différentier. Cependant, rien ne nous empêche d'utiliser un algorithme de CO à la place :



Comme nous l'expliquons dans le Chapitre 3, transformer une couche explicite en une couche implicite rend le travail de l'AD beaucoup plus difficile. Et même lorsque l'AD fonctionne, la solution d'un problème de CO est une fonction constante par morceaux, qui ne produit aucune dérivée utile.

Pour résoudre cette difficulté, nous publions un package Julia *open source* appelé `InferOpt.jl`, que nous décrivons dans le Chapitre 7. Cette librairie accepte n'importe quel algorithme de CO et l'approxime par une couche différentiable, ce qui assure la compatibilité avec l'écosystème AD et ML de Julia. Nous utilisons le concept unificateur de *couche CO probabiliste* pour englober un large éventail de méthodes présentes dans la littérature. Comme le montre la Figure 1.5a, une couche CO probabiliste convertit un problème d'optimisation linéaire en une distribution sur les sommets d'un polytope, qui varie continûment avec le vecteur objectif. Nous décrivons également de nouvelles techniques de différentiation (Propositions 7.3.4 et 7.3.5), dont les performances sont vérifiées empiriquement.

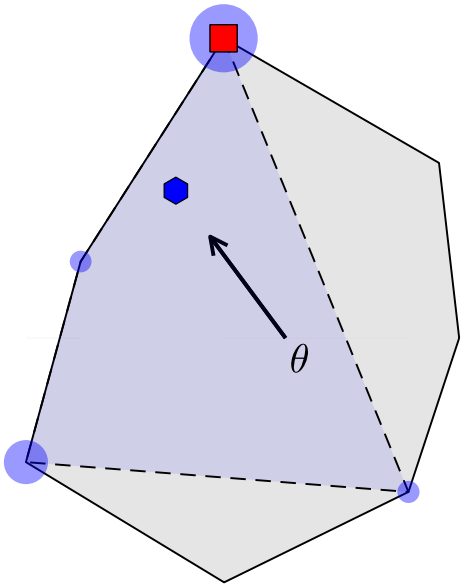
Ce travail est ensuite utilisé dans le Chapitre 11, où nous construisons un nouveau pipeline pour résoudre plus efficacement le problème d'allocation des voies. L'idée est d'apprendre où et pourquoi

les conflits sont susceptibles de se produire, pour les prévenir à l'avance au lieu de les réparer *a posteriori*.

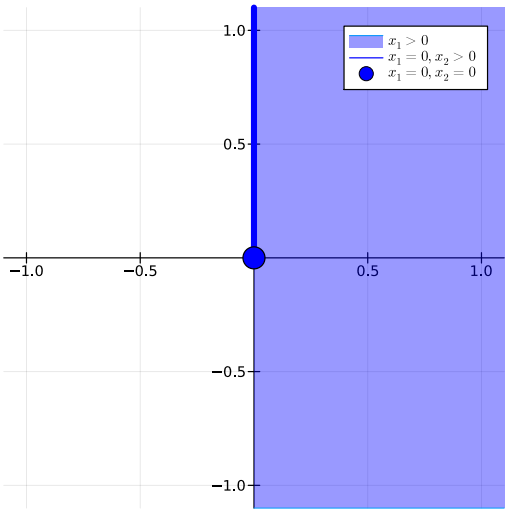
1.2.3.2 Généralisation à des objectifs multiples : le cas lexicographique

Bien qu'`InferOpt.jl` se concentre sur les problèmes de CO avec un objectif linéaire et un ensemble de solutions polyédral, ce ne sont pas les seuls problèmes intéressants. Par exemple, le Chapitre 5 montre que l'un des algorithmes les plus rapides pour le MAPF ne peut pas être formulé comme un programme linéaire en nombre entiers (ILP) avec un seul objectif. En revanche, nous pouvons le relier à un ILP multi-objectif, à condition de munir l'espace d'arrivée de l'ordre lexicographique. Pour insérer cet algorithme dans un pipeline d'apprentissage différentiable, la principale leçon du Chapitre 7 est qu'il nous faut une notion adéquate de convexité.

C'est pourquoi, dans le Chapitre 8, nous étendons la théorie de l'optimisation convexe au cadre lexicographique. En partant d'une définition de base de la convexité par rapport à l'ordre lexicographique, nous reconstruisons des objets bien connus comme les sous-gradients (Proposition 8.6.4) et les fonctions conjuguées. Toutefois, le cône lexicographique n'est pas fermé (voir Figure 1.5b), ce qui invalide plusieurs propriétés agréables du cas scalaire. Par exemple, les fonctions lexicographiquement convexes ne sont pas toujours continues (Exemple 8.4.13), et l'algorithme du sous-gradient lexicographique ne converge pas en général. En revanche, nous introduisons un algorithme de Jacobienne orthogonalisée (Algorithme 8.8.1) et une méthode de plans coupants lexicographiques (Algorithme 8.8.2), qui semblent plus prometteurs dans le cas général.



(a) Couche CO probabiliste donnée par une distribution sur les sommets d'un polytope (voir Chapitre 7)



(b) Le cône lexicographique $\mathcal{K} = \{x \in \mathbb{R}^2 : x \geq_{\text{lex}} 0\}$ n'est pas fermé (voir Chapitre 8)

Figure 1.5: Algorithmes d'optimisation dans les pipelines d'apprentissage

1.2.4 Logiciels *open source* pour une recherche reproductible

Dans la plupart des chapitres, nous réalisons des expériences numériques pour illustrer nos modèles et nos algorithmes. A une exception près, ces expériences sont toutes implémentées dans le langage de programmation Julia (Bezanson et al. 2017), un choix que nous justifions dans le Chapitre 3. Lorsque c'est possible, nous tâchons d'isoler les outils susceptibles d'être réutilisés, et nous publions un package *open source* les contenant.

Dans le Chapitre 9, notre modèle de prédiction des pannes est construit en empilant un HMM contrôlé sur un processus de Poisson. Par conséquent, nous fournissons les packages `ControlledHiddenMarkovModels.jl` et `PointProcesses.jl`, tous deux décrits au Chapitre 4. Le challenge Flatland du Chapitre 11 est abordé en combinant des algorithmes efficaces de recherche d'itinéraires avec des couches différentiables. Dans ce but, nous fournissons les packages `MultiAgentPathFinding.jl` (décrit au Chapitre 5) et `InferOpt.jl` (décrit au Chapitre 7). Enfin, l'une des techniques implémentées dans `InferOpt.jl` exploite le théorème des fonctions implicites, pour lequel nous avons développé le package `ImplicitDifferentiation.jl` (décrit au Chapitre 3).

Toutes les librairies mentionnées ci-dessus suivent les bonnes pratiques modernes en termes de développement logiciel. Leur code est disponible sur GitHub avec une licence *open source*. Il est accompagné d'une spécification claire des dépendances, et obéit à un système de versionnage sémantique. Après chaque nouvelle version, un processus d'intégration continue avec des tests unitaires garantit son bon fonctionnement. Mais surtout, chaque package possède une documentation détaillée, qui comprend dans certains cas des tutoriels pédagogiques.

Nous sommes convaincus que le respect de ces bonnes pratiques rend notre code plus fiable, et nos expériences plus facilement reproductibles. Par conséquent, les packages qui en résultent peuvent être considérés comme des produits de nos recherches, au même titre que la théorie qui les sous-tend.

2 Introduction

> Now, in terms of how we handle this moving forward, obviously, Earth is cancelled.
>
> *Judge Gen*
> The Good Place – S4E8
> The Funeral to End All Funerals (2019)

Contents

This thesis investigates the frontier between machine learning and combinatorial optimization, two active areas of applied mathematics research. We combine theoretical insights with efficient algorithms, and develop several open source Julia libraries. Inspired by a collaboration with the *Société nationale des chemins de fer français* (SNCF), we study high-impact use cases from the railway world: train failure prediction, delay propagation, and track allocation. However, we expect our work to be relevant in many other industrial contexts.

In the present chapter, we motivate our research with railway planning problems, before listing our main mathematical and computational contributions.

2.1 Industrial context: an overview of railway operations

2.1.1 Why study railways?

Droughts. Heatwaves. Wildfires. Thunderstorms. Floods. As climate change is spiraling out of control, reducing greenhouse gas emissions must become a top priority everywhere, especially in high-income countries. In France, the average individual's carbon footprint amounts to 9 tons of CO_2-equivalent per year, while the neutrality target for 2050 is set at 2 tons[1]. The changes necessary to reach that target will transform all aspects of our daily lives.

Currently, the transportation sector alone is responsible for about a third of all emissions, with road and air transport being the largest contributors. Thanks to their high capacity and energy efficiency (combined with a low-carbon electricity mix), French trains release much less CO_2 per kilometer and passenger than either cars or planes. Increasing the modal share of railways thus seems a worthy endeavor, but how can we entice more people to take the train? Of course, part of the answer lies with public policies and investment decisions, but the appeal of railways is not just about money and infrastructures. The quality of service also depends on fast and resilient decision-making, which is where applied mathematics comes into play.

2.1.2 Planning process and resources

Although travelers may not always realize it, moving a train from point A to point B is far from easy. Behind the scenes, a complex planning process unfolds, as described by Schlechte (2012, Chapter I). This process involves two main actors: railway companies (like SNCF *Voyageurs*), tasked with operating trains, and infrastructure managers (like SNCF *Réseau*), tasked with coordinating access to the network. Together, they strive to gather the resources needed for every journey.

Infrastructure is the defining resource of railway transport. Unlike cars and planes, trains reside on a very constrained network of tracks. Because braking distances are huge, they must maintain a large safety interval at all times to avoid collisions. Beyond low speeds, the safety interval often extends further than the driver can see. As a consequence, each trip has to book an itinerary in advance (*i.e.*, a timed sequence of track sections) that is provably free from conflict. Since it would be wasteful to preempt a full itinerary for the entire duration of the journey, individual track sections are claimed and released as trains go by. This allows safety intervals to be respected, while ensuring high throughput.

[1] https://www.statistiques.developpement-durable.gouv.fr/edition-numerique/chiffres-cles-du-climat-2022/

Rolling stock and *crew* are other essential resources. A given trip can only take place with a vehicle that satisfies specific requirements (capacity, number of coaches, engine type), staffed with adequate personnel (driver, ticket inspector, snack salesperson). Last, but not least, *passengers* can also be considered a resource, since trains are useless (and costly) when traveling empty. Departures are often delayed to make connections possible for passengers, just like they can be delayed to wait for an available vehicle or driver.

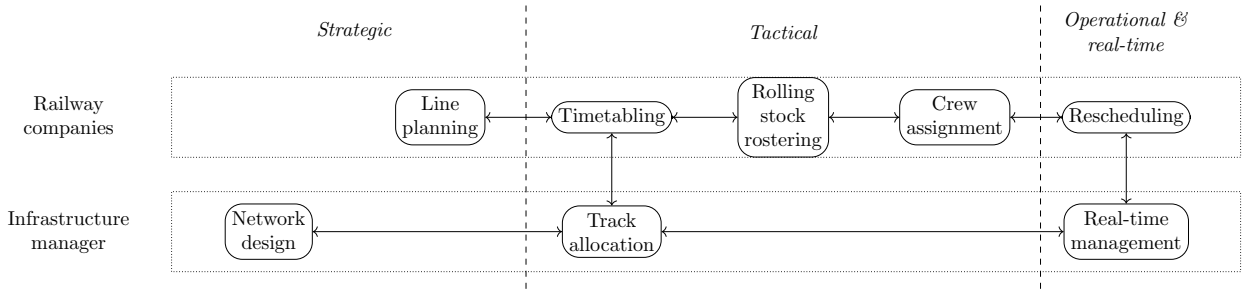


Figure 2.1: Main phases of railway planning
Adapted from Schlechte (2012) with kind permission from the author

Railway planning stretches over several time scales. At the *strategic* level (several years in advance), structural decisions like network design and line planning are made. Then, at the *tactical* level (several months in advance), timetable construction and track allocation take place in parallel. This is a necessary condition for rolling stock rostering and crew assignment to happen. Next, at the *operational* level (several days in advance), last-minute adjustments are made to the schedule. Finally, *real-time* perturbations to traffic are monitored and handled by dedicated control centers. These phases are summarized on Figure 2.1.

2.1.3 Optimization and learning for a resilient system

At each stage of the planning process, railway companies and infrastructure managers must answer a hard question: what is the best possible use for our resources? This decision involves many variables to set, several constraints to satisfy, as well as a cost function to minimize, which is why we call it an *optimization problem*. To make matters worse, this problem is *combinatorial*: it has a finite but exponentially large set of possible solutions, so that exhaustive enumeration becomes intractable. The field of Combinatorial Optimization (CO) is dedicated to the study of such problems, and it has given rise to many efficient algorithms since the 1950s (Korte and Vygen 2006). Applications of CO to railways are almost as old as the field itself, and they are still very relevant today (Borndörfer et al. 2018).

However, CO algorithms are not always sufficient when used out of the box. First, standard problem formulations assume that the future holds no surprise, and that everything proceeds according to plan. Yet history shows that incidents happen, so we need to anticipate them during the scheduling phase. Second, most optimization routines are designed to work on a broad variety of instances. But in real life, we are interested in one specific use case (say, one railway network), and we would like our solution procedure to perform well on this use case, even if it means giving up on generality. Machine Learning (ML) is the key to overcoming both of these hurdles. It encompasses a large family of methods designed to extract valuable information from raw data (Murphy 2012).

And luckily, railways generate a lot of data: from departure and arrival times to train condition monitoring or passenger counts, measurements abound. The real challenge lies in making sense of this data, before exploiting it within optimization algorithms.

2.1.4 Three important railway problems

In this thesis, our ultimate goal is to perform *track allocation*. We want to take a timetable as input, and construct a feasible itinerary for each train through the network. To generate realistic networks and timetables, we rely on the simulator of the Flatland challenge (Mohanty et al. 2020), whose output is displayed on Figure 2.2. Track allocation for Flatland is a special case of Multi-Agent Path Finding (MAPF), where a set of agents must find non-conflicting paths through a graph and reach their destination as fast as possible.

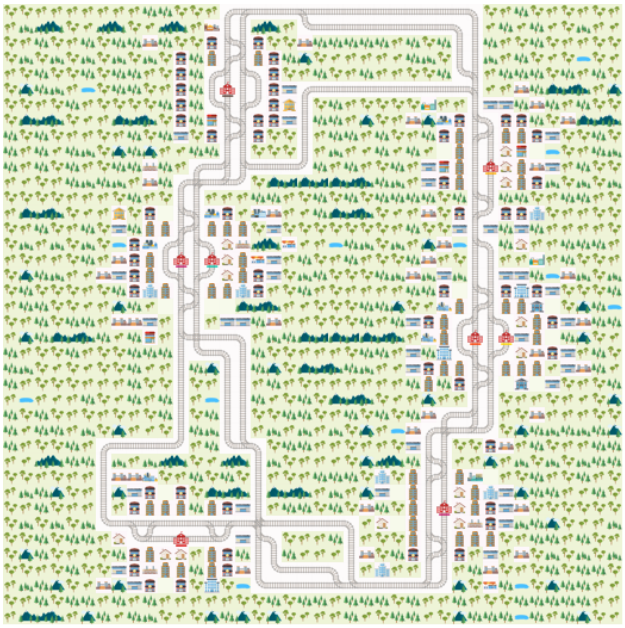


Figure 2.2: Example map from the Flatland challenge (see Chapter 11)

Chapter 5 discusses MAPF in more detail. As we will see, it is an **NP**-hard problem, whose main solution methods rely on decomposition techniques, polynomial approximations or local search. We adapt and implement a large neighborhood search algorithm that remains fast even on very large instances, and we test it on a popular benchmark set. Next, in Chapter 11, we describe a way to improve this algorithm by learning from simulated Flatland data.

Chapters 5 and 11 focus on the deterministic version of MAPF / track allocation. But what happens if we also need to face unpredictable, stochastic delays? Then, we need statistical models to feed our planning routines with trustworthy predictions.

As illustrated by Figure 2.3, train delays are split between two categories: primary delays and secondary delays. Primary delays are exogenous: they are caused by incidents such as mechanical malfunctions, bad weather or unexpected human behavior. Conversely, secondary delays are endogenous: they are generated by interactions between trains on the network. To achieve resilient track allocation, we must learn to predict both kinds of delays.

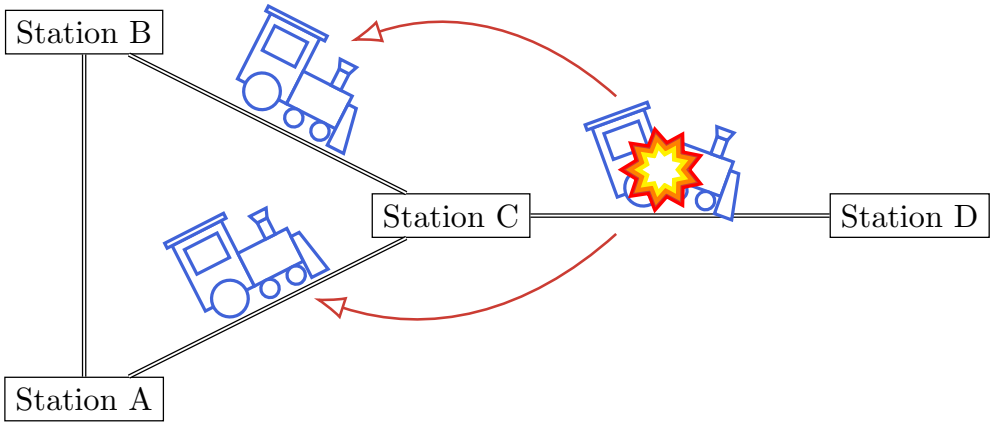


Figure 2.3: Two kinds of delays: primary (explosion $\implies$ failure) and secondary (arrow $\implies$ propagation)

Rolling stock failures are among the main causes of primary delays, which is why we investigate *failure prediction* in Chapter 9. Based on condition monitoring data from a fleet of regional trains, we try to anticipate their degradation and predict their next stop at the repair workshop. In future work, this information will help us identify the most vulnerable vehicles, and assign them to the safest itineraries.

The snowball effect giving rise to secondary delays is called *delay propagation*, and we study it in Chapter 10. To quantify this phenomenon, our method does not rely on confidential industrial data, only departure and arrival times. We apply it to actual records from a Swiss tramway system, and recover coherent insights about the dynamics of track congestion. In future work, this information will help us identify the most critical parts of the network, so that we can avoid putting too much load on them.

2.2 Scientific contributions: fast data-driven decision-making

Before delving into railway applications, we use Chapters 3 through 8 to develop theoretical and algorithmic tools for data-driven decision-making. Combining ML and CO means we face the following challenges:

1. How to extract meaningful information from high-dimensional and noisy data?
2. How to use this information to enhance optimization algorithms?

We now list our main contributions to each of these research topics.

2.2.1 High-dimensional learning with hidden variables

The first recurring theme in our work is the notion of hidden variable. It comes up in failure prediction (Chapter 9) to model the health state of a train, and then again in delay propagation (Chapter 10) to capture the notion of network congestion. In both of these settings, we work with temporal models of the generic form

$$X_t = f_t(X_{t-1}) \qquad \text{and} \qquad Y_t = g_t(X_t) \tag{2.1}$$

Here, X_t is the latent process, which we do not have access to, while Y_t is the observation process. The functions f_t and g_t are random: we call them the transition and emission distributions respectively. In some cases, finding a suitable parametric form for f_t and g_t is far from obvious. In other cases, the model is fairly simple, but its estimation raises deep statistical questions.

2.2.1.1 Modeling complex data streams

In the context of failure prediction (Chapter 9), our main challenge is to construct an adequate degradation model from the data we are given. Train activity logs are made available once a day: we regard them as control variables that influence the transition and emission distributions. On the other hand, our observations consist of a stream of events recorded in continuous time, which is not compatible with a discrete time formulation. To make things worse, both the events and the controls are represented by high-dimensional vectors, parts of which are possibly useless or redundant.

To overcome these obstacles, we build upon the concepts reviewed in Chapter 4 to define a *controlled Hidden Markov Model* with *marked Poisson process observations*. This model is illustrated on Figure 2.4a: it merges processes in discrete and continuous time without needing to compress information. The transition and emission distributions are specifically designed to reflect our knowledge of the train deterioration process. Careful hypotheses on the stochastic dependencies help reduce the number of learnable parameters, thus alleviating the curse of dimensionality.

2.2.1.2 Obtaining statistical guarantees from sparsity

In the context of delay propagation (Chapter 10), we introduce a novel framework centered around the congestion variable X_t. This congestion moves through the network, influencing departure and arrival times Y_t as it makes track sections more difficult to cross. To model X_t (the latent process) and Y_t (the observation process), we use a *partially-observed Vector AutoRegression*:

$$X_t = \theta X_{t-1} + \varepsilon_t \qquad \text{and} \qquad Y_t = \Pi_t X_t + \eta_t \tag{2.2}$$

In Equation (2.2), ε_t and η_t are white Gaussian noise processes with respective variances σ^2 and ω^2. Meanwhile, Π_t is a random mask that hides each component of Y_t with probability $1-p$, possibly displaying temporal correlations. Finally, θ is a sparse matrix of transition weights, with no more than s non-zero coefficients per row. Estimating θ is key to understanding delay dynamics, which is why we need to control the estimation error.

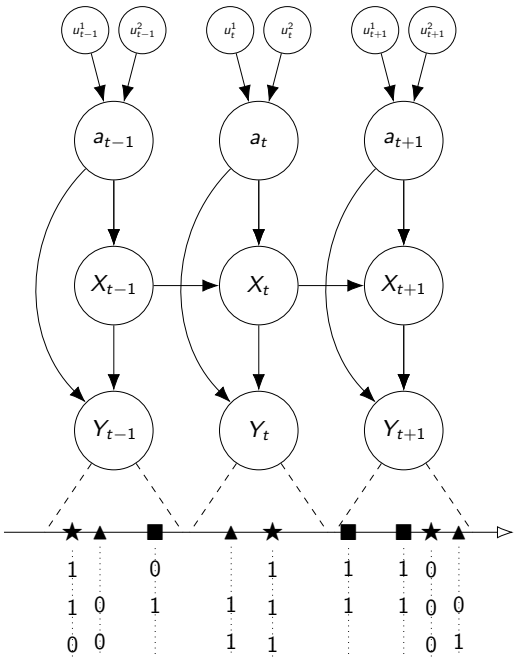
Chapter 6 provides a complete analysis of this slightly unusual state-space model. By deriving a new upper bound (Theorem 6.4.1) and the associated minimax lower bound (Theorem 6.4.2), we formally quantify the error of the optimal estimator on a sequence of length T:

$$\|\widehat{\theta} - \theta\|_\infty \propto \left(1 + \frac{\omega^2}{\sigma^2}\right) \frac{s}{p\sqrt{T}}$$

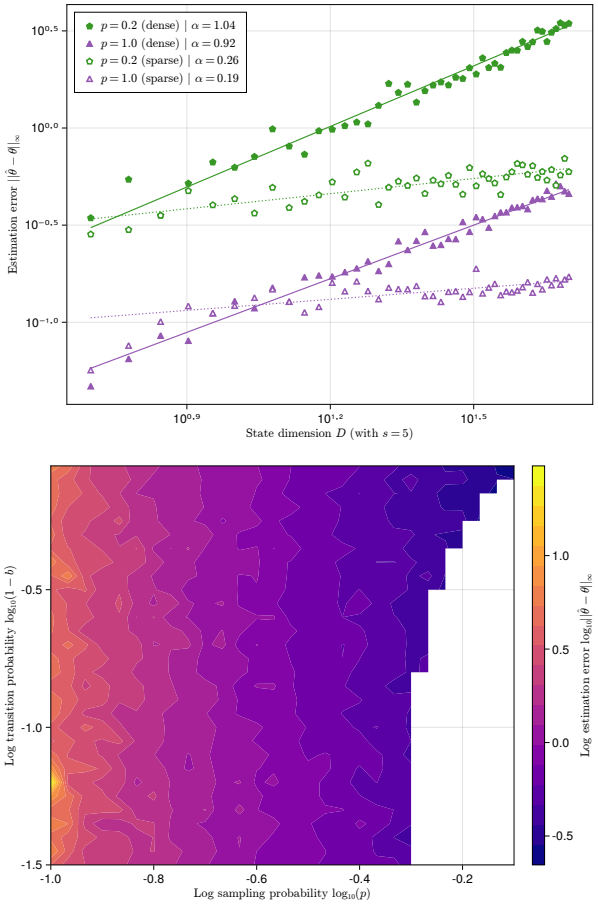
Unsurprisingly, transition sparsity (low s) makes estimation easier, while observation sparsity (low p) makes estimation harder, as shown on Figure 2.4b.

2.2.2 Optimization algorithms within learning pipelines

The second recurring theme in our work is the concept of differentiable learning pipeline.



(a) Controlled HMM + Poisson process = a hierarchical model for failure prediction (see Chapter 9)

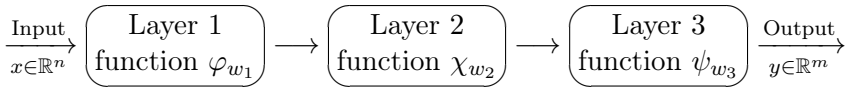


(b) Estimation of a partially observed VAR is easier with small s and large p (see Chapter 6)

Figure 2.4: High-dimensional learning with hidden variables

2.2.3 What is a pipeline?

These last few years, deep learning has achieved impressive results for natural language processing, image generation, and many other tasks that were once thought to be out of reach (Goodfellow, Bengio, and Courville 2016). Neural networks work by applying a succession of basic operations such as linear combinations or nonlinear activation functions. These operations are called *layers*, and together they form a (directed acyclic) computational graph, also called a *pipeline*:



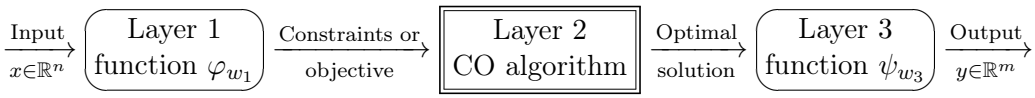
Each of the layers comes with internal parameters, or *weights*, which need to be selected during the training phase. Most of the time, training involves minimizing a loss function $\mathcal{L}_w$ defined on N (input, output) samples:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_w(x^{(i)}, \overline{y}^{(i)}).$$

Typically, the loss measures the distance between the output of the network applied to input x and the target output $\overline{y}$. It is usually minimized with some variant of gradient descent, but working out derivatives manually becomes very tedious for complex pipelines. That is where Automatic Differentiation (AD) comes in: it computes numerically exact derivatives for each layer without needing an analytical formula. Then, AD combines these derivatives with the chain rule (see Chapter 3) to allow seamless backpropagation of loss gradients onto the layer weights.

2.2.3.1 Differentiable combinatorial layers

Most layers from ML are fairly simple and explicit functions, which are easy to differentiate. However, nothing stops us from using a CO algorithm instead:



As we explain in Chapter 3, turning an explicit layer into an implicit one makes the job of AD much more difficult. And even if AD still works, the solution of a CO problem is a piecewise constant function, so no useful derivative information can come from it.

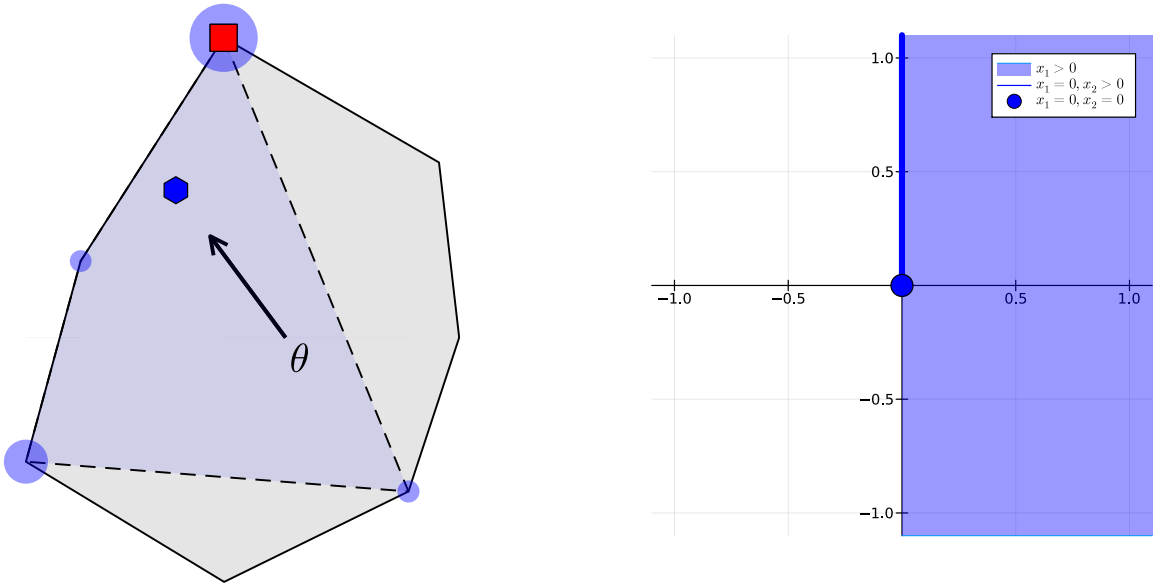
To address this difficulty, we publish an open source Julia package called `InferOpt.jl`, which we describe in Chapter 7. Our library takes any CO algorithm and constructs an approximate differentiable layer, making it compatible with most AD and ML frameworks in the Julia ecosystem. We leverage the unifying concept of *probabilistic CO layer* to encompass a wide array of previous methods. As displayed on Figure 2.5a, a probabilistic CO layer converts a linear optimization problem into a distribution on the vertices of a polytope, which changes smoothly with the objective vector. We also describe new differentiation techniques (Propositions 7.3.4 and 7.3.5), whose performance is demonstrated empirically.

Our package is then used in Chapter 11, where we design several pipelines to solve MAPF more efficiently. The basic idea is to learn where and why conflicts are likely to happen, so that we can prevent them beforehand instead of repairing them afterwards.

2.2.3.2 Generalization to multiple objectives: the lexicographic case

While `InferOpt.jl` focuses on CO problems with a linear objective and polyhedral feasible set, these are not the only problems of interest. For instance, Chapter 5 shows that one of the fastest algorithms for MAPF cannot be formulated as a single-objective Integer Linear Program (ILP). Instead, we can relate it to a multiobjective ILP, as long as we endow the objective space with the lexicographic order. To insert this algorithm into a differentiable learning pipeline, the main takeaway of Chapter 7 is that we need an adequate notion of convexity.

That is why, in Chapter 8, we extend the theory of convex optimization to the lexicographic setting. Starting from a basic definition of convexity with respect to the lexicographic order, we reconstruct well-known objects like convex subgradients (Proposition 8.6.4) and conjugates. But because the lexicographic cone is not closed (see Figure 2.5b), several aspects from the real-valued case fail to generalize. For example, lexicographically-convex functions are not always continuous (Example 8.4.13), and the lexicographic subgradient algorithm usually fails to converge. On the other hand, we introduce an orthogonalized Jacobian algorithm (Algorithm 8.8.1) and a method of lexicographic cutting planes (Algorithm 8.8.2), which seem more promising in the general case.



(a) Probabilistic CO layer given by a distribution on polytope vertices (see Chapter 7)

(b) The lexicographic cone $\mathcal{K} = \{x \in \mathbb{R}^2 : x \geq_{\text{lex}} 0\}$ is not closed (see Chapter 8)

Figure 2.5: Optimization algorithms within learning pipelines

2.2.4 Open source software for reproducible research

In most chapters, we perform numerical experiments to illustrate our models and algorithms. Bar one exception, these experiments are always implemented in the Julia programming language (Bezanson et al. 2017), a choice we justify in Chapter 3. Whenever possible, we abstract away the parts that can be reused by other people, and publish an open source package containing them.

Our failure prediction model from Chapter 9 is built by stacking a controlled HMM on top of

a Poisson process. Hence, we provide the packages `ControlledHiddenMarkovModels.jl` and `PointProcesses.jl`, both described in Chapter 4. Tackling the Flatland challenge in Chapter 11 involves wrapping efficient MAPF algorithms inside differentiable layers. To this end, we provide the packages `MultiAgentPathFinding.jl` (described in Chapter 5) and `InferOpt.jl` (described in Chapter 7). Finally, one of the layers in `InferOpt.jl` exploits the implicit function theorem, for which we develop the package `ImplicitDifferentiation.jl` (described in Chapter 3).

All the libraries mentioned above follow modern best practices in terms of software development. Their source code is available on GitHub with a permissive open source license. They contain precise requirement lists and follow semantic versioning. After each commit, a continuous integration process with unit tests ensures correctness. And most importantly, they come with extensive documentation, which includes (in some cases) pedagogic tutorials for end users.

We trust that abiding by these best practices makes our academic code more trustworthy, and our experiments more easily reproducible. Therefore, the resulting packages can be considered valuable outputs of our research, just as much as the theory behind them.

2.3 Outline of the dissertation

2.3.1 Main parts

The present document is divided into three parts.

- Part I describes some important algorithms and the software packages that we developed for them. Since most of these algorithms are not new, our main contribution is a performant, generic and reliable Julia implementation.
- Part II is the mathematical core of our work. It contains theoretical contributions related to statistics, optimization and their interactions.
- Part III deals with railway applications. It leverages the previous chapters to make sense of real data and improve decision-making processes.

2.3.2 Chapter dependencies

We sum up the structure of the thesis on Figure 2.6. To interpret this directed acyclic graph, just apply the following principle recursively: one should not read a chapter until one has read every one of its parents.

2.3.3 Experiments

All experiments (except those in Chapter 7) were performed on a Dell Precision 5530 mobile workstation with Intel Core i7-8850H CPU (2.60GHz $\times$ 12) and 31 GiB of RAM, running under Ubuntu 20.04.

2.4 Notations

We denote by $\mathbb{N}$ the set of natural numbers, by $\mathbb{Z}$ the set of integers, by $\mathbb{Q}$ the set of rational numbers, and by $\mathbb{R}$ the set of real numbers. Real intervals are written $[a, b)$, where a bracket means that the

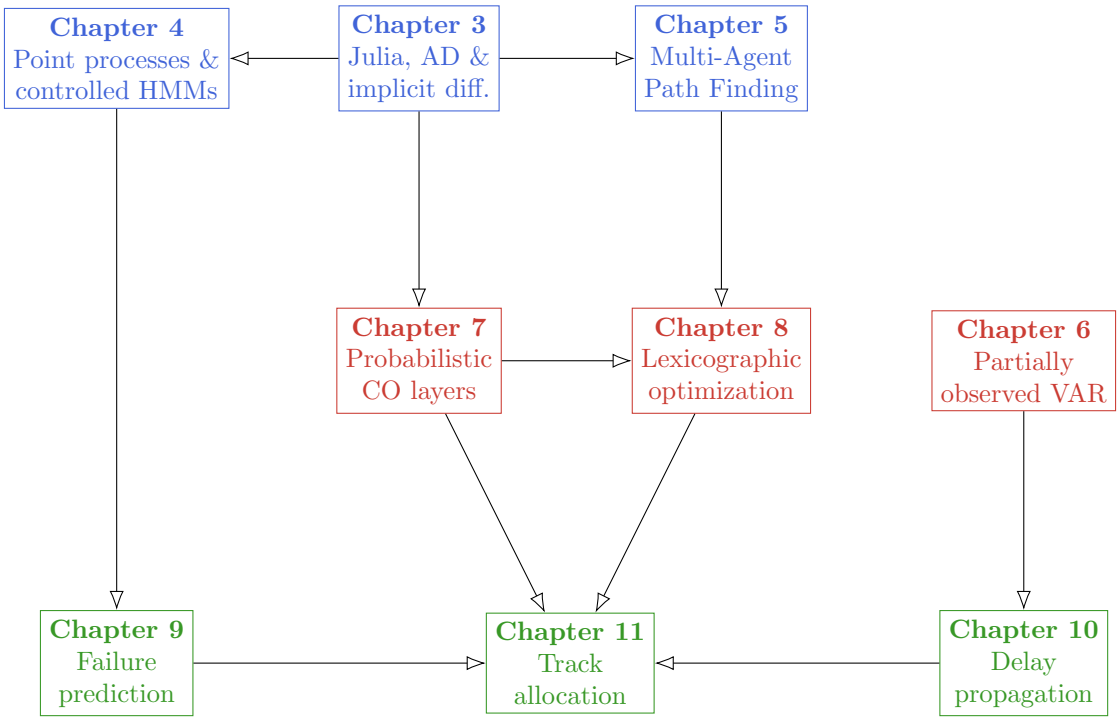


Figure 2.6: Dependencies between chapters
Part I – Part II – Part III

bound is included, and a parenthesis means that the bound is excluded. As an example, we define the closed real half-line $\mathbb{R}_+ = [0, +\infty)$. We also extend the real line by defining $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. Integer intervals are written $[\![i, j]\!]$, where both bounds are always included. As a shortcut, we often write $[n]$ for $[\![1, n]\!]$. Given a real number $x \in \mathbb{R}$, we denote by $|x|$ its absolute value. When we write $\log(x)$, we mean the natural logarithm with base e.

2.4.1 Linear algebra

A vector $x \in \mathbb{R}^d$ with components x_i is written $x = (x_1, \ldots, x_d)$. We denote by $\|x\|_2$ its Euclidean norm, by $\|x\|_1$ its ℓ_1 norm, by $\|x\|_\infty$ its ℓ_∞ norm, and by $\|x\|_0$ its number of nonzero entries. The notation e_i stands for the basis vector with a 1 at position i and zeros elsewhere, while $\mathbf{1}$ is the vector with all components equal to 1. Integer intervals can be used to select subvectors, in which case we shorten $[\![i, j]\!]$ into $i{:}j$. For instance, $x_{i:j} = (x_i, ..., x_j)$. If $x, y \in \mathbb{R}^d$, we denote by $x \leq y$ the componentwise order (which is a partial order), and by $x \leq_{\text{lex}} y$ the lexicographic order (which is a total order).

A real matrix $M \in \mathbb{R}^{d \times d}$ can be defined using its coefficients $M = (M_{i,j})_{1 \leq i,j \leq d}$ or its columns $M = \begin{pmatrix} M_1 & \cdots & M_d \end{pmatrix}$. We denote by $M^\top$ its transpose, by M^{-1} its inverse and by $M^\dagger$ its Moore-Penrose pseudo-inverse. Its trace is written $\mathrm{Tr}(M)$ and its determinant $\det(M)$. We denote its singular values by $\varsigma_{\max}(M) = \varsigma_1(M) \geq \varsigma_2(M) \geq \cdots \geq \varsigma_d(M) = \varsigma_{\min}(M)$. We denote its eigenvalues by $\lambda_1(M), \ldots, \lambda_d(M)$, and whenever they all belong to $\mathbb{R}$ (which is notably the case for real symmetric matrices), we order them as $\lambda_{\max}(M) = \lambda_1(M) \geq \lambda_2(M) \geq \cdots \geq \lambda_d(M) = \lambda_{\min}(M)$.

The column-wise flattening of matrix M into a vector is written $\mathrm{vec}(M)$.

We will need to work with several norms and related quantities. The operator ℓ_1 norm $\|M\|_1 = \sup_{x\neq 0} \frac{\|Mx\|_1}{\|x\|_1} = \max_j \sum_i |M_{i,j}|$ is the maximum ℓ_1 norm of a column of M. The operator ℓ_∞ norm $\|M\|_\infty = \sup_{x\neq 0} \frac{\|Mx\|_\infty}{\|x\|_\infty} = \max_i \sum_j |M_{i,j}|$ is the maximum ℓ_1 norm of a row of M. The operator ℓ_2 norm $\|M\|_2 = \sup_{x\neq 0} \frac{\|Mx\|_2}{\|x\|_2} = |\varsigma_{\max}(M)| = \sqrt{\lambda_{\max}(M^\top M)}$ is also known as the spectral norm. The Frobenius norm $\|M\|_F = \|\mathrm{vec}(M)\|_2 = \sqrt{\mathrm{Tr}(M^\top M)}$ is the Euclidean norm of the flattened matrix. The maximum eigenvalue of the entries $\|M\|_{\max} = \|\mathrm{vec}(M)\|_\infty = \max_{i,j}|M_{i,j}|$ and the spectral radius $\rho(M) = \max_i |\lambda_i(M)|$ will also be useful.

Given two real matrices A and B with compatible sizes, we denote by AB their usual product, by $A \odot B$ their Hadamard (elementwise) product, and by $A \otimes B$ their Kronecker (tensor) product. The Loewner order on symmetric matrices is defined by $A \preceq B$ if and only if $B - A$ has only nonnegative eigenvalues, it is a partial order.

We write I for the identity matrix, and J_r for the square matrix entirely filled with zeros, except for the subdiagonal of rank r which is filled with ones. The notation $\mathrm{diag}(\lambda)$ stands for a diagonal matrix with coefficients $\lambda_1, ..., \lambda_d$. We will sometimes need to introduce block matrices, in which case we will use the special notation $M = (M_{[b_1,b_2]})_{b_1,b_2}$ instead of $M = (M_{i,j})_{i,j}$. We define $\mathrm{bdiag}_k(M)$ as a block-diagonal matrix with k copies of M on the diagonal and zeros elsewhere.

When we want to apply a function elementwise, we often use notation that is standard for real numbers but not for matrices: for instance, $\sqrt{M} = (\sqrt{M_{i,j}})_{i,j}$ and $\frac{1}{M} = (\frac{1}{M_{i,j}})_{i,j}$. When we use integer intervals to select submatrices, we mean a selection with respect to columns: $M_{i:j} = \begin{pmatrix} M_i & \cdots & M_j \end{pmatrix}$.

2.4.2 Probability

We denote by $\mathbb{P}$ a probability distribution or probability density function. For a random variable X, let $\mathbb{E}[X]$ be its expectation. If X takes real values, we denote its variance by $\mathrm{Var}(X)$. If X takes vector values, we denote its covariance matrix by $\mathrm{Cov}(X)$. Similarly, the covariance (matrix) between variables X and Y is written $\mathrm{Cov}(X, Y)$. The notations $\mathbb{1}_E$ and $\mathbb{1}\{E\}$ stand for the indicator function of an event E. We write $\mathrm{KL}\left\{\mathbb{P}_1 \parallel \mathbb{P}_2\right\}$ for the Kullback-Leibler (KL) divergence between two probability distributions. We often make use of several standard distributions:

- Discrete: the Bernoulli distribution $\mathcal{B}(p)$, the binomial distribution $\mathcal{B}(n, p)$, the multinomial distribution $\mathcal{M}(p)$, the Poisson distribution $\mathcal{P}(\lambda)$.
- Continuous: the uniform distribution $\mathcal{U}(a, b)$, the exponential distribution $\mathcal{E}(\lambda)$, the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$.

2.4.3 Analysis

We use $\Delta^d = \left\{p \in \mathbb{R}^d : p_i \geq 0, \sum_i p_i = 1\right\}$ to refer to the unit simplex of dimension d, and $\mathcal{B}(x, r)$ for the Euclidean ball of center x and radius r. If $\mathcal{S} \subset \mathbb{R}^d$ is a set, we define its convex hull as $\mathrm{conv}(\mathcal{S}) = \left\{\sum_i p_i x_i : d \in \mathbb{N}, x \in \mathcal{S}^d, p \in \Delta^d\right\}$. The orthogonal projection onto a set $\mathcal{S}$ (when it exists) is written $\mathrm{proj}_{\mathcal{S}}$, while the orthogonal projection onto a vector u is written proj_u.

If f is a real-valued function, we denote by $\nabla_p f(x)$ the gradient of f with respect to parameter p at point x, and by $\partial_p f(x)$ its convex subdifferential (set of subgradients). Its Hessian matrix is written $\mathrm{H}_p f(x)$. The notation $\mathrm{dom}(f)$ stands for the domain of f, *i.e.*, the set on which

it takes finite values. If $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is a vector-valued function, we denote by $\mathrm{J}_p f(x)$ its Jacobian matrix with respect to parameter p. The function f can be expressed using its real-valued components $f_j\colon x \in \mathbb{R}^n \longmapsto f_j(x)$. The notation $f_{|\mathcal{X}}$ stands for the restriction of any function f to a subset $\mathcal{X}$ of its input space.

2.4.4 Frequent symbols

Whenever possible, we try to give the following letters a coherent meaning throughout the thesis. An undirected graph will usually be called $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertices u or v and edges e. We also use the same notation for directed graphs, even though edges $e \in \mathcal{E}$ should normally be replaced by arcs $a \in \mathcal{A}$ (one reason is that the letter a already refers to "agents" in several chapters). Time is always represented by letters such as t or τ. A dimension will often be called d, n or m. Parameters (in the statistical sense) are named θ or w depending on the context. A random variable called X_t is usually a hidden state, while Y_t is an observation.

Part I

Algorithms and open source packages

3 Julia for scientific computing and automatic differentiation

> I am fast. To give you a reference point, I'm somewhere between a snake and a mongoose... and a panther.
>
> *Dwight Schrute*
> The Office – S3E8
> The Merger (2006)

Contents

In this chapter, we justify our decision to use the Julia language for nearly all of our numerical experiments. We then survey Automatic Differentiation (AD), an essential ingredient of modern Machine Learning (ML). As learning pipelines become more complex, AD calculates and combines derivatives without requiring user input or introducing numerical errors. It will play a crucial role in Chapters 4 and 7. Finally, we discuss the implicit function theorem and its application to building implicit layers around numerical solvers. A generic implementation is provided in our open source package `ImplicitDifferentiation.jl`[1].

The Julia package described here is the result of a collaboration with Mohamed Tarek, from Pumas AI / University of Sydney. This chapter is partly based on a conference talk we gave at JuliaCon 2022 (D. and Tarek 2022).

3.1 Julia for scientific computing

The *two-language problem* is an infamous, albeit unwritten, law of scientific computing. While programmers can easily build prototypes in high-level dynamic languages (Python, R, Matlab, *etc.*), they often need to switch to low-level static languages (C, C++, Fortran, *etc.*) when speed becomes an issue. This makes development of high-performance code more cumbersome and less accessible.

The Julia language is a recent addition to the scientific computing ecosystem: it was publicly announced in 2012, and the 1.0 release only came out in 2018. It attempts to tackle the two-language problem through careful design choices, which we review below. Our main source is the official Julia article by Bezanson et al. (2017), as well as the technical documentation of the language[2].

3.1.1 Performance benchmarks

Before delving into language technicalities, we briefly mention the Computer Language Benchmarks Game[3], an open source comparison of many modern programming languages. It is built around 10 computationally heavy tasks, ranging from differential equations (N-body problem) to linear algebra (spectral norm) or high-precision arithmetic (digits of π). For every language, implementations were submitted and improved by the GitHub community, before being applied to instances of various sizes. The goal was to reach the best possible performance according to several criteria.

Here, we only consider the fastest run on the largest instance of each task. We use the CPU time as measure of speed, and the size of the compressed source as a proxy for code complexity. Figure 3.1 displays the geometric means of these values over all tasks, for each language. As we can see, Julia is located on the Pareto front (bottom left), which means it strikes an interesting balance between efficiency and simplicity.

For a deeper investigation of various performance measures, we refer the reader to Pereira et al. (2017). In particular, the authors discuss the relative merits of programming languages with respect to electricity consumption, which is a key aspect of sustainable scientific computing. Their latest results include Julia[4], which ranks second only to Rust in terms of energy efficiency.

[1] https://github.com/gdalle/ImplicitDifferentiation.jl
[2] https://docs.julialang.org/en/v1/
[3] https://benchmarksgame-team.pages.debian.net/benchmarksgame/
[4] https://sites.google.com/view/energy-efficiency-languages/updated-functional-results-2020#h.p_Ai6_HV718M7v

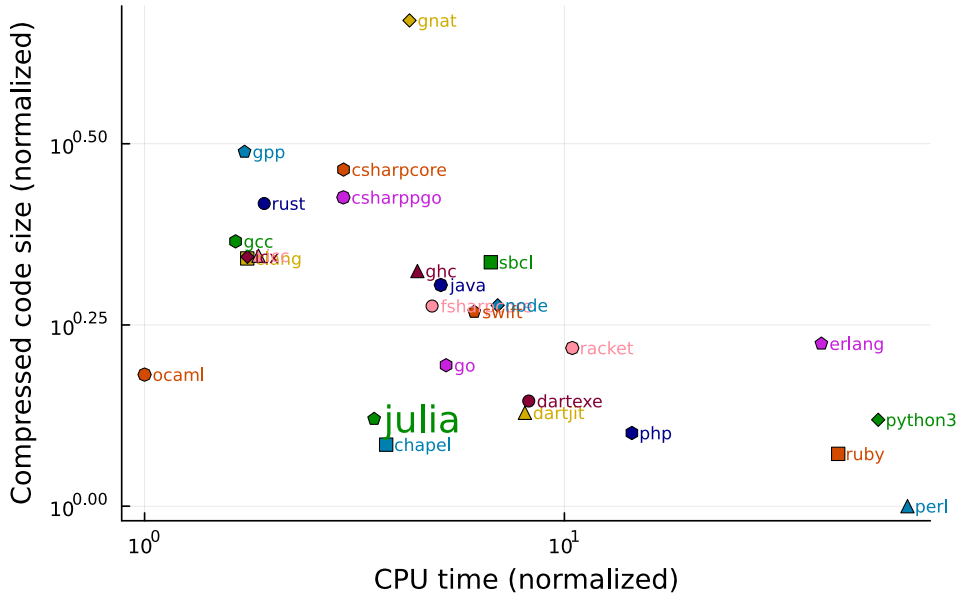


Figure 3.1: A compromise between code simplicity and efficiency (data from the Computer Language Benchmarks Game)

3.1.2 Types and multiple dispatch

Julia's central tenet is its type system, which works by combining specialization and abstraction. When a chunk of code is run, the type inference algorithm determines the type of every function input, and propagates it recursively to every function output. Then, *multiple dispatch* selects a specialized method for each function and subfunction, based on inferred argument types. If this method does not exist yet, it is compiled just-in-time. The more specialized the method, the faster it will be, because knowing the memory layout of the data greatly improves the generated machine code.

For instance, the built-in + function has over 200 specialized methods in the standard library, allowing addition between various types of numbers (integer, floating point, complex, rational) but also arrays (vectors, matrices, tensors) and even time periods. And while + works correctly for any pair of abstract matrices, more efficient implementations exist that leverage sparsity, symmetry, or any other structure that can be encoded in the matrix type.

3.1.3 Composability

Thanks to the type inference algorithm, type annotations are mostly optional. They are used for code selection and do not affect performance, except in ambiguous cases. The reason behind the name "multiple dispatch" is that the selection procedure considers every argument of a function, not just the first one. Methods are not owned by the objects they apply to, as opposed to what happens in object-oriented languages.

It is not uncommon for the developer of a package A to borrow a type from package B, another one from package C, and implement a new function that takes an argument of each type. Packages

B and C do not need to be aware of package A for this approach to be efficient. Multiple dispatch thus provides Julia with a high potential for *composability*, and also explains the fragmented nature of the ecosystem. If we want to combine, say, neural networks with integer programming solvers, there is no need for a single package containing both. Instead, we can blend existing packages for deep learning and discrete optimization, using just a few lines of independent glue code. In this specific example, the glue code would be our `InferOpt.jl` library (see Chapter 7).

3.1.4 Generic programming

The key requirement for composability is *generic programming*. When they write high-level functions, Julia programmers are encouraged to only make minimal assumptions about the argument types. Genericity does not impact performance, because for any given function input, a specialized method will be compiled that relies on efficient low-level operations. On the other hand, genericity allows downstream users to explore innovative applications of a package, which upstream developers may not have predicted beforehand.

A classic example consists in producing code that accept any subtype of `Number`, not just `Float64`. The Julia ecosystem has many number types, each of them with a specific purpose:

- standard single- and double-precision floating point numbers (`Float32` and `Float64`)
- arbitrary precision floating point numbers (`BigFloat`)
- dual numbers for AD (see `ForwardDiff.jl`[5])
- logarithmic-scale numbers to avoid numerical over- and underflow (see `LogarithmicNumbers.jl`[6])
- quantities with units (see `Unitful.jl`[7]) or uncertainty bars (see `Measurements.jl`[8])
- real intervals for exact floating point arithmetic (see `IntervalArithmetic.jl`[9])

Our own Julia packages take great care to remain as generic as possible, so as not to limit the range of possible uses. For example, `ImplicitDifferentiation.jl` (see below) and `InferOpt.jl` both accept arbitrary callable objects to solve optimization problems, without prescribing a specific modeling framework. `PointProcesses.jl` and `ControlledHiddenMarkovModels.jl` (see Chapter 4) only interact with probability measures through the functions `rand` (for simulation) and `logdensityof` (for likelihood computation). This makes them compatible with a wide variety of distributions, and with each other. In addition, they work with arbitrary number types. Finally, `MultiAgentPathFinding.jl` (see Chapter 5) is designed to accept any kind of graph object, which means custom implementations can be plugged in for special cases like grid graphs.

[5] https://github.com/JuliaDiff/ForwardDiff.jl
[6] https://github.com/cjdoris/LogarithmicNumbers.jl
[7] https://github.com/PainterQubits/Unitful.jl
[8] https://github.com/JuliaPhysics/Measurements.jl
[9] https://github.com/JuliaIntervals/IntervalArithmetic.jl

3.1.5 Performance guidelines

While Julia programs are fairly easy to write, it does take some practice to make them reach their top speed. The main performance tips can be summed up as follows: facilitate type inference and reduce memory allocations. When type inference fails, the compiler produces inefficient machine code, because it has to anticipate every possible argument type instead of focusing on one. And even within fully type-inferrable functions, memory allocation and garbage collection are significant bottlenecks. Luckily, inference failures and excessive allocations can be diagnosed with dedicated introspection tools like the built-in code profiler.

Our goal here is not to give detailed recipes for producing good Julia code. We only wish to highlight that it is non-trivial, and that each one of our own packages was written, profiled and optimized with this goal in mind. For instance, because `ImplicitDifferentiation.jl` and `InferOpt.jl` are optimizer-agnostic, making type inference successful requires special attention. `ControlledHiddenMarkovModels.jl` takes care to re-use storage and leverage efficient linear algebra routines, while also remaining type-inferrable. `MultiAgentPathFinding.jl` benefits from a reflection on the right priority queues for Dijkstra-like algorithms. These are but a few examples, and more details will be given in the relevant chapters.

3.2 Automatic differentiation

We now move on to AD, also called algorithmic differentiation, which deduces numerically exact derivatives from the source code of a function. A good reference on this topic is given by Griewank and Walther (2008), while uses of AD in ML are explored by Baydin et al. (2018). Our exposition is mostly borrowed from Margossian (2019).

3.2.1 Numerical differentiation methods

In the field of numerical optimization, we frequently need to compute derivatives of a function $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$. If we do not want to work out these derivatives by hand, we have three main options:

- *Symbolic differentiation* uses computer algebra to convert a mathematical expression for f into a mathematical expression for $\partial f / \partial x_i$.
- *Finite differentiation* approximates derivatives with a difference between function values at neighboring points:
$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \varepsilon \mathrm{e}_i) - f(x)}{\varepsilon}$$
- *Algorithmic differentiation* re-interprets the code that computes the function f, and combines derivatives from individual building blocks thanks to the chain rule.

Symbolic differentiation is less error-prone than manual derivation, but the number of terms in the resulting formulas can blow up exponentially. This often happens when repeated products or compositions are involved. Furthermore, its main focus is a class of functions given by mathematical expressions. Conversely, finite differentiation and AD both support a wide variety of computer programs, including those with control flow such as `if` statements or `while` loops.

In finite differentiation, the scale of ε dictates the precision of the Taylor expansion. Unfortunately, numerical cancellation errors can occur when ε becomes too small. In addition, computing gradients that way requires $\Theta(n)$ function evaluations, since we need to cycle through each basis vector e_i. On the other hand, AD provides exact derivatives wherever they exist. Even better, reverse mode AD only requires one function call to compute a gradient. This is especially interesting for ML, where we must often differentiate loss functions with respect to high-dimensional parameters. In the rest of this chapter, we only focus on AD: see Baydin et al. (2018) and the references therein for details on symbolic and finite differentiation.

3.2.2 Forward and reverse mode AD

AD comes in two flavors: forward and reverse mode. Following Margossian (2019), we illustrate them by considering the case of a composite function with L layers:

$$f = f^L \circ f^{L-1} \circ \cdots \circ f^2 \circ f^1 : \mathbb{R}^n \longrightarrow \mathbb{R}^m$$

3.2.2.1 Chain rule and products

In both cases, the core ingredient is the chain rule of differentiation, which expresses the full Jacobian $\mathrm{J}f(x)$ as a matrix product:

$$J = J^L J^{L-1} \cdots J^2 J^1 \quad \text{with} \quad J^\ell = \mathrm{J}f^\ell \left[(f^{\ell-1} \circ \cdots \circ f^1)(x) \right] \tag{3.1}$$

In Equation (3.1), note the difference between J, which denotes a specific matrix, and J, which denotes the Jacobian operator. Given an *input perturbation* (or tangent) $u \in \mathbb{R}^n$, forward mode AD is concerned with computing the *Jacobian-vector product* Ju. It does so by accumulating products in the forward order, from layer 1 to layer L:

$$Ju = J^L(J^{L-1} \cdots (J^2(J^1 u))) \tag{3.2}$$

Given an *output sensitivity* (or adjoint) $v \in \mathbb{R}^m$, reverse mode AD is concerned with computing the *vector-Jacobian product* $v^\top J$. It does so by accumulating products in the reverse order, from layer L to layer 1:

$$v^\top J = (((v^\top J^L) J^{L-1}) \cdots J^2) J^1 \tag{3.3}$$

With a computation budget roughly proportional to one call of f, forward mode AD yields one Jacobian-vector product, while reverse mode AD yields one vector-Jacobian product (Griewank and Walther 2008, Chapter 4). Suppose we want the full Jacobian matrix instead. Forward mode computes it column-by-column (taking $u = \mathrm{e}_i$ for $i \in [n]$), whereas reverse mode computes it row-by-row (taking $v = \mathrm{e}_j$ for $j \in [m]$). The former is a good choice when $n \ll m$, but the latter shines when $n \gg m$. In particular, real-valued functions satisfy $m = 1$, and we can compute their gradient with a single vector-Jacobian product $1^\top \mathrm{J}f(x) = \nabla f(x)^\top$.

3.2.2.2 Implementation details

Each mode of AD comes with its own constraints and limitations, summed up in Table 3.1. Forward mode AD often works by introducing *dual numbers* of the form $x + u\epsilon$, where ϵ is an abstract number satisfying $\epsilon \neq 0$ but $\epsilon^2 = 0$. We can think of ϵ as representing an order 1 quantity in

	Forward mode	**Reverse mode**
Propagated quantity	Input perturbation	Output sensitivity
Implementation	Operations on dual numbers	Additional storage + backward pass
Jacobian complexity	$O(n)$ function calls	$O(m)$ function calls

Table 3.1: Comparison between forward and reverse mode AD for $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$

a Taylor expansion, so that dual numbers encode both a value and its derivative at the same time. For this trick to work, every layer f^ℓ must accept inputs of dual type, with the following behavior: $f^\ell(x + u\epsilon) = f^\ell(x) + J^\ell u\epsilon$. We recover the Jacobian-vector product by feeding $x + u\epsilon$ to f and querying the coefficient of ϵ in the final result. Importantly, as soon as a layer is applied, its input can be safely discarded because its output contains all the necessary information. This keeps memory requirements quite light.

Reverse mode AD is usually more complicated to implement, not least because it works in two stages. First, a forward sweep, which serves to compute the intermediate values $(f^{\ell-1} \circ \cdots \circ f^1)(x)$ from Equation (3.1). Second, a backward sweep, during which vector-Jacobian products are formed and propagated, hence the name *backpropagation*. Reverse mode AD uses much more memory: because the reverse sweep only starts when the forward sweep is complete, the intermediate values must be stored in the meantime. For more complex pipelines, exploring and remembering the directed acyclic computation graph also generates overhead. Reducing memory requirements of reverse mode AD is the subject of ongoing research.

3.2.3 Julia ecosystem

Since the Julia programming language was designed for scientific computing, it is no surprise that AD plays an important role in its package ecosystem. Many AD frameworks have been developed over the years, and Schäfer et al. (2021) offer a good summary of their respective strengths and weaknesses. Useful information can also be found on the web page of the JuliaDiff GitHub organization[10]. It is essential to keep in mind that most AD frameworks fall short of supporting every single language construct. Each project that we list comes with its own trade-offs in terms of breadth and performance.

Regarding forward mode AD, `ForwardDiff.jl`[11] (Revels, Lubin, and Papamarkou 2016) is as close as it gets to a community standard. It works on any code that is generic enough to handle inputs of abstract number type `Real` (which includes dual numbers). For reverse mode AD, there are more options, such as `ReverseDiff.jl`[12] or `Zygote.jl`[13] (Innes 2019). Among these libraries, a frequent limitation is the inability to differentiate code that mutates arrays. This is unfortunate, because in-place modifications are often synonymous with reduced runtime, as we mentioned above. Note however that the experimental project called `Enzyme.jl`[14] (W. Moses and Churavy 2020; W. S. Moses et al. 2021) seeks to remove this limitation.

[10] https://juliadiff.org/
[11] https://github.com/JuliaDiff/ForwardDiff.jl
[12] https://github.com/JuliaDiff/ReverseDiff.jl
[13] https://github.com/FluxML/Zygote.jl
[14] https://github.com/EnzymeAD/Enzyme.jl

The `ChainRules.jl`[15] package (White et al. 2022) aims at creating a common interface for AD in Julia. It is built around the concept of forward and reverse rules, *i.e.*, routines that compute Jacobian-vector and vector-Jacobian products. Coming back to Equation (3.1), defining a forward rule for f^ℓ means writing a function $u \longmapsto J^\ell u$, while defining a reverse rule means writing a function $v \longmapsto v^\top J^\ell$. Once these chain rules are defined, they are compatible with many AD frameworks (including `Zygote.jl` and `ReverseDiff.jl`). The associated functions can thus be used as layers in most ML packages, like the `Flux.jl`[16] library for neural networks (Innes 2018). All it takes is for the user to define f as the composition of L layers $f^1, \dots, f^L$, and AD will seamlessly chain individual derivatives by applying Equation (3.3).

`ChainRules.jl` already contains rules for most of the Julia standard library, which is more than enough for basic applications. However, users can also provide custom rules for a function they programmed themselves. Custom rules are useful when the function in question:

- Is not AD-compatible due to black box components (*e.g.*, calls to an external solver): we explore this case below.
- Is not AD-compatible due to a limitation of the AD framework (*e.g.*, array mutation): we give a concrete example in Chapter 4.
- May be AD-compatible but does not have useful derivatives (*e.g.*, because it is piecewise constant): we discuss this situation at length in Chapter 7.

We refer the reader to the `ChainRules.jl` documentation[17] for more information on both the mathematical and algorithmic aspects. We also mention Kochenderfer and Wheeler (2019, Chapter 2), in which elementary Julia implementations can be found for forward mode AD.

3.3 Implicit differentiation

3.3.1 The limits of automatic differentiation

Scientific computing often deals with functions that are given by a complex iterative solver. Common examples include differential equations, optimization problems, fixed point equations, nonlinear systems, *etc.* It is natural to ask whether we can compute their derivatives easily using AD. Alas, our solver may not be compatible with the AD library we want to use. For example, many industrial solvers are implemented in lower-level programming languages, which Julia AD tools cannot make sense of. And even when the solver is fully amenable to AD, differentiating through an iterative procedure is very expensive. It requires unrolling the iterations and propagating derivatives through each one. This can make storage requirements explode in reverse mode, because the space required in memory is roughly proportional to the number of operations.

3.3.2 The implicit function theorem

Implicit differentiation provides a solution to this problem, by specifying what we expect from a solution *regardless of the solver we use to compute it.* Our exposition is mostly drawn from Blondel,

[15] https://github.com/JuliaDiff/ChainRules.jl
[16] https://github.com/FluxML/Flux.jl
[17] https://juliadiff.org/ChainRulesCore.jl/stable/

Berthet, et al. (2022), but another good resource is the tutorial by Kolter, Duvenaud, and M. Johnson (2020). Consider a function $f\colon x \in \mathbb{R}^n \longmapsto y = f(x) \in \mathbb{R}^m$. While the solver computing f may be a black box, we are often able to characterize its output using a set of *conditions*. For instance, the solution to an unconstrained differentiable minimization problem $f(x) = \operatorname{argmin}_{y \in \mathbb{R}^m} g(x, y)$ will always satisfy gradient stationarity $\nabla_1 g(x, f(x)) = 0$. More generally, we define the conditions as a mapping $C\colon (x, y) \in \mathbb{R}^n \times \mathbb{R}^m \longmapsto C(x, y) \in \mathbb{R}^m$ such that for all $x \in \mathbb{R}^n$, we have

$$C(x, f(x)) = 0 \tag{3.4}$$

Let us see what happens when we differentiate the previous equation with respect to x:

$$\underbrace{\mathrm{J}_1 C(x, f(x))}_{B} + \underbrace{\mathrm{J}_2 C(x, f(x))}_{-A} \underbrace{\mathrm{J}f(x)}_{J} = 0 \tag{3.5}$$

We obtain a linear system linking the Jacobian $\mathrm{J}f(x) \in \mathbb{R}^{m \times n}$ to the partial Jacobians $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{m \times n}$ of the conditions C. Provided that A is nonsingular, we can invert the linear system and obtain the Jacobian $\mathrm{J}f(x)$ we are interested in. This is, in essence, the meaning of the *implicit function theorem*. To recap, implicit differentiation involves 3 steps:

1. Use any solver to compute $f(x)$.
2. Apply AD on Equation (3.4) to deduce A and B.
3. Solve the system $AJ = B$ to recover $\mathrm{J}f(x)$.

3.4 The package `ImplicitDifferentiation.jl`

3.4.1 Main ideas

While implicit differentiation is already available in Python via the `jaxopt`[18] package of Blondel, Berthet, et al. (2022), Julia implementations are hard to come by. Our package `ImplicitDifferentiation.jl`, developed with Mohamed Tarek, aims at filling this gap. It is inspired by a prior attempt from `NonconvexUtils.jl`[19], but strives to make the code more simple and user-friendly.

`ImplicitDifferentiation.jl` is designed to be fully compatible with the `ChainRules.jl` specification. Users only need to define f and C as Julia functions, and wrap both of them inside an `ImplicitFunction` callable object. When we call this object outside an AD pipeline, it simply returns $f(x)$. But when we try to differentiate through it, implicit differentiation overrides the default unrolling of iterations thanks to custom forward and reverse rules. Our package includes some implementation tricks to improve numerical efficiency. These ideas are already exposed by Blondel, Berthet, et al. (2022) among others, so we did not invent anything. Instead, our main contribution consists in finding the right design and combination of tools, which allowed us to turn these ideas into code.

[18] https://github.com/google/jaxopt
[19] https://github.com/JuliaNonconvex/NonconvexUtils.jl

3.4.2 Implementation details

Remember from Equations (3.2) and (3.3) that we are not interested in manipulating the full Jacobian matrix if we can avoid it. Luckily, from Equation (3.5), we can compute Jacobian-vector products based on the observation that for any $u \in \mathbb{R}^n$, we have $A(Ju) = Bu$. Thus, we only need to solve $Ap = Bu$ for $p = Ju$. On the other hand, vector-Jacobian products are obtained for any $v \in \mathbb{R}^m$ by seeking $w \in \mathbb{R}^m$ such that $A^\top w = v$. It follows directly that $v^\top J = w^\top AJ = w^\top B$. Since we can dispense with storing the full Jacobian matrix J, we do not want to store A and B either. The solution is to consider them as lazy linear operators, *i.e.*, functions $v \longmapsto Av$ and $v \longmapsto Bv$ that are evaluated on the fly.

We use `LinearOperators.jl`[20] (Orban and Siqueira 2019) to express A and B. Of course, we also need a linear system solver that is compatible with lazy operators, and it turns out that Krylov subspace methods satisfy this requirement. Therefore, we use the package `Krylov.jl`[21] (Orban 2019) to solve linear systems within the forward and reverse rules. We select GMRES (Saad and Schultz 1986) as the default algorithm, but the user is free to specify another one if additional information is available on the properties of A (like symmetry).

3.4.3 Numerical experiments

Despite its recent release, downstream uses of our package are already starting to emerge. For instance, `DifferentiableFactorizations.jl`[22] is a lightweight toolbox which relies on `ImplicitDifferentiation.jl` to differentiate through many types of matrix factorizations: (generalized) eigenvalue decomposition, singular value decomposition, Cholesky factorization, LU factorization, QR factorization. Indeed, matrix factorizations are typical cases where the solver may be complicated, but the conditions satisfied by the output are simple.

We conclude this chapter by presenting a more sophisticated application of `ImplicitDifferentiation.jl` to optimal transport. The reader can refer to Peyré and Cuturi (2019) for a thorough introduction to this topic.

3.4.3.1 Optimal transport with entropic regularization

Suppose we have a distribution of mass $a \in \Delta^n$ over points $x_1, \dots, x_n \in \mathbb{R}^p$ (where Δ denotes the probability simplex). We want to transport it to a distribution $b \in \Delta^m$ over points $y_1, \dots, y_m \in \mathbb{R}^p$. The unit moving cost from point x to point y is proportional to the squared Euclidean distance $d(x, y) = \|x - y\|_2^2$.

A transportation plan can be described by a coupling $p \in \Pi(a, b)$, i.e. a probability distribution on the product space with the right marginals:

$$\Pi(a, b) = \{p \in \Delta^{n \times m} : p\mathbf{1} = a \text{ and } p^\top \mathbf{1} = b\}$$

Let $D \in \mathbb{R}^{n \times m}$ be the distance matrix, with $D_{ij} = d(x_i, y_j)$. The basic optimization problem we want to solve is a linear program:

$$p(D) = \operatorname*{argmin}_{p \in \Pi(a,b)} \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} D_{ij}$$

[20] https://github.com/JuliaSmoothOptimizers/LinearOperators.jl
[21] https://github.com/JuliaSmoothOptimizers/Krylov.jl
[22] https://github.com/mohamed82008/DifferentiableFactorizations.jl

```julia
function sinkhorn(K; a, b, T)
    u, v = a, b
    for t in 1:T
        u = a ./ (K * v)
        v = b ./ (K' * u)
    end
    return u
end
```

Code sample 3.1: Sinkhorn algorithm

```julia
function fixed_point(K, u; a, b)
    v = b ./ (K' * u)
    difference = u .- a ./ (K * v)
    return difference
end
```

Code sample 3.2: Sinkhorn fixed point condition

```julia
sinkhorn_implicit = ImplicitFunction(sinkhorn, fixed_point)

function transportation_plan(D; a, b, eps, T)
    K = exp.(.-D ./ eps)
    u = sinkhorn_implicit(K; a=a, b=b, T=T)  # or u = sinkhorn(K; a=a, b=b, T=T)
    v = b ./ (K' * u)
    p = u .* K .* v'
    return p
end;
```

Code sample 3.3: Using a differentiable Sinkhorn wrapper

In order to make it smoother, we add an entropic regularization term with scaling $\varepsilon > 0$:

$$p_\varepsilon(D) = \underset{p \in \Pi(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(p_{ij} D_{ij} + \varepsilon p_{ij} \log \frac{p_{ij}}{a_i b_j} \right)$$

3.4.3.2 The Sinkhorn algorithm

To solve the regularized problem, we can use the Sinkhorn fixed point algorithm. Let $K \in \mathbb{R}^{n \times m}$ be the matrix defined by $K_{ij} = \exp(-D_{ij}/\varepsilon)$. Then the optimal coupling $p_\varepsilon(D)$ can be written as:

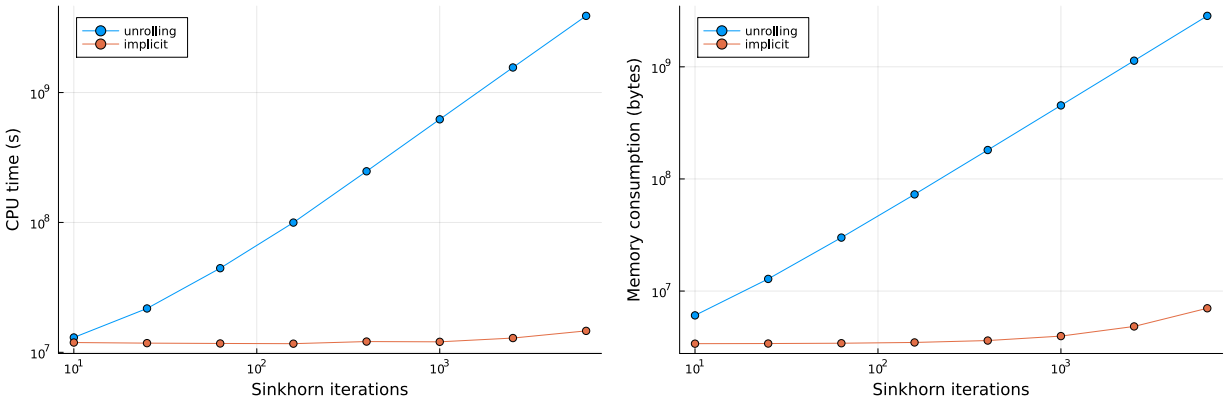
$$p_\varepsilon(D) = \operatorname{diag}(u) K \operatorname{diag}(v)$$

where u and v are the fixed points of the following Sinkhorn iteration:

$$u^{t+1} = \frac{a}{Kv^t} \qquad \text{and} \qquad v^{t+1} = \frac{b}{K^\top u^t} \tag{3.6}$$

The corresponding Julia function is given by Code sample 3.1. Equation (3.6) does not only give us an actionable algorithm: it also yields the fixed point condition that the output of said algorithm must satisfy. We show the relevant Julia function in Code sample 3.2.

Using our package, it is straightforward to construct an `ImplicitFunction` object wrapping the Sinkhorn iteration: we simply combine the function f (Code sample 3.1) with the conditions C (Code sample 3.2). We can then use this callable and differentiable object when computing the optimal transportation plan, as shown on Code sample 3.3.



(a) Implicit differentiation saves CPU time

(b) Implicit differentiation saves memory

Figure 3.2: Implicit differentiation vs. iteration unrolling: the case of the Sinkhorn algorithm

3.4.3.3 Time and memory impact of implicit differentiation

Our numerical setup involves point clouds of size $n = m = 10$, sampled uniformly at random in the unit hypercube with dimension $p = 100$, and a regularization parameter set to $\varepsilon = 1$. Our goal is to highlight the performance benefits of implicit differentiation compared to naive unrolling. We thus measure computation time and memory use for both approaches, while varying the number of Sinkhorn iterations T from 10 to 10000.

Results are presented on Figure 3.2. As expected, the CPU time necessary for unrolling increases linearly with T (Figure 3.2a), mainly driven by the growing storage requirements of reverse mode AD (Figure 3.2b). On the other hand, our implicit wrapper removes the need for unrolling, and its performance remains stable even as the number of iterations becomes large. Our experiment thus validates the relevance of `ImplicitDifferentiation.jl` as a user-friendly take on the implicit function theorem, which enables efficient derivative computations.

4 Temporal point processes & controlled Hidden Markov Models

People assume that time is a strict progression of cause to effect, but actually – from a non-linear, non-subjective viewpoint – it's more like a big ball of wibbly-wobbly... timey-wimey... stuff.

The Doctor
Doctor Who – S3E10
Blink (2007)

Contents

In this chapter, we briefly review temporal point processes and controlled Hidden Markov Models (HMMs). These stochastic processes have a wide variety of applications, but so far generic Julia implementations have been lacking. That is why we introduce two open source Julia packages called `PointProcesses.jl`[1] and `ControlledHiddenMarkovModels.jl`[2], which contain many useful algorithms for modeling, simulation and learning. We will combine both packages to construct our failure prediction framework in Chapter 9.

4.1 Temporal point processes

Point processes are probabilistic models for collections of events. Their general theory is presented in great detail by Daley and Vere-Jones (2003). Here we restrict ourselves to point processes on the real half-line, hence the adjective "temporal". Our exposition is mostly borrowed from J. G. Rasmussen (2018).

4.1.1 Events and marks

In a marked temporal point process, each event is a couple (t, m), where $t \in \mathbb{R}_+$ is the event time and $m \in \mathfrak{M}$ is the event mark (which belongs to an arbitrary set). The realization of a marked temporal point process is a possibly infinite sequence of events $y = \{(t^k, m^k) : k \in \mathbb{N}\}$. However, we usually observe it on a bounded time interval $[t_{\min}, t_{\max})$, during which only a finite number K of events occur. Figure 4.1 displays an example observation sequence.

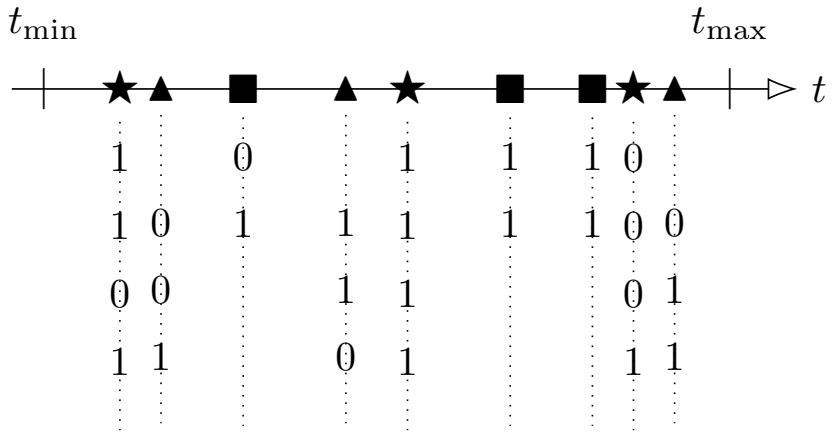


Figure 4.1: A realization of a temporal point process with mark set $\mathfrak{M} = \{\bigstar, \blacktriangle, \blacksquare\} \times \{0, 1, \varnothing\}^4$

4.1.2 Intensity function

A marked temporal point process is defined by its *conditional intensity function* $\lambda^*(t, m)$, which depends on the time $t \in \mathbb{R}_+$ and the mark $m \in \mathfrak{M}$. As is customary in the literature, we write λ^*

[1] https://github.com/gdalle/PointProcesses.jl
[2] https://github.com/gdalle/ControlledHiddenMarkovModels.jl

to indicate that the intensity is conditioned on the history of the process up to (but not including) time t. The conditional intensity quantifies the instantaneous rate of occurrence of a specific event given everything that has happened before. For a given $m \in \mathfrak{M}$, within a small time interval of length $\mathrm{d}t$ around $t \in \mathbb{R}_+$, one expects to observe $\lambda^*(t, m)\mathrm{d}t$ events with mark m. The conditional intensity can be written as a product $\lambda^*(t, m) = \lambda_g^*(t)p^*(t, m)$, where

- the *ground intensity* $\lambda_g^*(t) = \sum_{m \in \mathfrak{M}} \lambda^*(t, m)$ is the instantaneous rate of occurrence for all marks combined;

- the *mark distribution* $p^*(t, m) = \lambda^*(t, m)/\lambda_g^*(t)$ gives the probability of observing mark m, conditioned on the fact that an event occurs at time t.

4.1.3 Simulation

To simulate marked temporal point processes, a generic method is Ogata's modified thinning algorithm (Ogata 1981), which is a form of rejection sampling. It uses a local upper bound $B(t)$ on the ground intensity, which is only valid on the interval $[t, t + L(t))$, to generate a candidate next event. Then, it decides whether to keep or reject it by comparing the upper bound to the real ground intensity. We give the pseudocode in Algorithm 1.

Algorithm 1: Simulation of a temporal point process (Ogata's algorithm)

$t \leftarrow t_{\min}$, $k \leftarrow 0$
while $t < t_{\max}$ **do**
  Compute $B(t)$ and $L(t)$
  Draw $\Delta \sim \mathcal{E}(B(t))$
  if $t + \Delta > t + L(t)$ **then**
    $t \leftarrow t + L(t)$
  else
    $t \leftarrow t + \Delta$
    Draw $U \sim \mathcal{U}(0, 1)$
    if $\frac{\lambda_g^*(t)}{B(t)} < U$ **then**
      $k \leftarrow k + 1$
      Draw $m \sim p^*(t, \cdot)$
      $(t^k, m^k) \leftarrow (t, m)$
    end
  end
end
Return $\{(t^k, m^k)\}$

As in rejection sampling, when the upper bound becomes looser, the simulation becomes more costly.

4.1.4 Learning

The likelihood of a sequence of events $y = \{(t^k, m^k) : k \in [K]\}$ on interval $[t_{\min}, t_{\max})$ is given by

$$\mathbb{P}(y) = \underbrace{\left[\prod_{k=1}^{K} \lambda^*(t^k, m^k)\right]}_{\text{individual events}} \underbrace{\exp\left(-\int_{t_{\min}}^{t_{\max}} \lambda_g^*(t)\ \mathrm{d}t\right)}_{\text{empty time between events}} \tag{4.1}$$

Which yields the following log-likelihood:

$$\log \mathbb{P}(y) = \sum_{k=1}^{K} \log \lambda^*(t^k, m^k) - \int_{t_{\min}}^{t_{\max}} \lambda_g^*(t)\ \mathrm{d}t \tag{4.2}$$

If the conditional intensity function has a parametric form, then its parameters can be estimated by maximizing the log-likelihood. While explicit formulas exist in a few simple cases, numerical optimization is usually necessary as soon as the conditional intensity becomes more complex.

4.1.5 Poisson processes

To simplify computations, it is tempting to assume that the intensity function is unconditional, *i.e.*, that it does not depend upon the past history. This simpler model is called a *Poisson process*. If we further assume that the intensity function is homogeneous (does not depend upon the current time), then we are left with a homogeneous Poisson process, for which $\lambda^*(t, m) = \lambda(m)$. This makes simulation much easier and faster, as demonstrated by Algorithm 2.

Algorithm 2: Simulation of a homogeneous Poisson process

```
Draw K ~ P(λ_g)
for k = 1, ..., K do
    Draw t^k ~ U(t_min, t_max)
    Draw m^k ~ λ(·)/λ_g
end
Return {(t^k, m^k)} (sorted by increasing t^k)
```

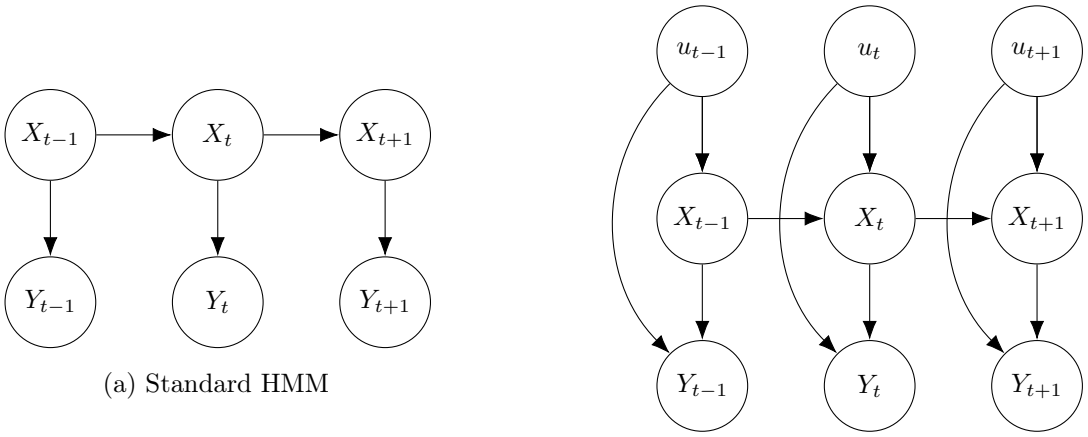
In this case, Equation (4.2) boils down to

$$\log \mathbb{P}(y) = \sum_{k=1}^{K} \log(\lambda(m^k)) - (t_{\max} - t_{\min}) \sum_{m \in \mathfrak{M}} \lambda(m)$$

As a consequence, the Maximum Likelihood Estimator (MLE) is explicit:

$$\widehat{\lambda}(m) = \frac{\text{count}(y, m)}{\text{duration}(y)} \quad \text{where} \quad \begin{cases} \text{count}(y, m) & = |\{k \in [K] : m^k = m\}| \\ \text{duration}(y) & = t_{\max} - t_{\min} \end{cases}$$

Cox processes are an extension of Poisson processes, whose intensity $\lambda(X_t, m)$ depends on an underlying stochastic process X_t. They are also known as doubly-stochastic Poisson processes. We will give an example in a moment, when we discuss HMMs with Poisson process emissions.



(a) Standard HMM

(b) Controlled HMM

Figure 4.2: Graphical model structures

4.2 Hidden Markov Models with control variables

HMMs are a class of discrete-time stochastic processes with latent variables. The book by Cappé, Moulines, and Rydén (2005) provides a thorough treatment of related models and algorithms. Here, we mainly draw inspiration from the concise tutorial by Rabiner (1989).

4.2.1 Hidden Markov Models and control variables

A standard HMM is composed of two stochastic processes indexed by a time $t \in \mathbb{N}$: the *state* X_t, which is hidden, and the *emissions* Y_t, which are observed. The state takes discrete values in $[S]$ and evolves according to a Markov chain, with transition matrix P and initial distribution p. Meanwhile, the emission distribution is chosen according to the current value of the state. This dependency structure is represented on Figure 4.2a, and summed up as follows:

$$X_1 \sim p, \quad X_t \sim \mathbb{P}(X_t \mid X_{t-1}) \qquad \text{and} \qquad Y_t \sim \mathbb{P}(Y_t \mid X_t)$$

A controlled HMM comprises an additional layer of *controls* u_t, which are assumed to be deterministic (and therefore predictable). In our setting, control variables can influence both the evolution of the state and the distribution of emissions. In particular, neither of these processes is stationary anymore. This dependency structure is represented on Figure 4.2b, and summed up as follows:

$$X_1 \sim p, \quad X_t \sim \mathbb{P}(X_t \mid X_{t-1}, u_t) \qquad \text{and} \qquad Y_t \sim \mathbb{P}(Y_t \mid X_t, u_t)$$

This is similar in spirit to the Input-Output HMM proposed by Bengio and Frasconi (1994), but they assume more parameter independence than we do (between transitions and emissions, and between each state). Note that the standard HMM is nothing but a special case with empty controls.

4.2.2 Statistical algorithms

We now describe the statistical problems related to controlled HMMs, along with their solutions. Suppose we are given a sequence of observations $y_1, \dots, y_T$ and the associated controls $u_1, \dots, u_T$.

Let us denote by θ a generic parameter for our model. There are three main tasks we want to perform:

1. Inference: given θ, compute the posterior distribution of the past states $X_1, \dots, X_T$

$$\mathbb{P}_\theta(X_t = x_t \mid Y_{1:T} = y_{1:T}, u_{1:T})$$

2. Prediction: given θ, compute the posterior distribution of the future states $X_{T+1}, \dots, X_{T+h}$

$$\mathbb{P}_\theta(X_{T+h} = x_{T+h} \mid Y_{1:T} = y_{1:T}, u_{1:T+h})$$

3. Estimation: find the parameter θ that maximizes the likelihood of the observations

$$\widehat{\theta} \in \operatorname{argmax} \mathbb{P}_\theta(Y_{1:T} = y_{1:T} \mid u_{1:T})$$

From now on, to simplify notations, we keep the dependence on θ implicit. We also omit conditioning on the sequence of controls u_t, since it is deterministic.

4.2.2.1 Inference

The first task can be tackled using the forward-backward algorithm, initially proposed by Baum et al. (1970). If we try to compute the posterior state distribution naively, Bayes' formula requires summing over all possible state trajectories $x_{1:T}$. That is of course intractable, which is why the forward-backward algorithm uses dynamic programming to compute posteriors efficiently. It exploits the forward and backward variables, defined as follows:

$$\alpha_t(i) = \mathbb{P}(Y_{1:t} = y_{1:t}, X_t = i) \qquad \beta_t(i) = \mathbb{P}(Y_{t+1:T} = y_{t+1:T} \mid X_t = i)$$

Let us define the vector of emission densities $e_t \in [0,1]^S$ at time t, such that

$$e_t(i) = \mathbb{P}(Y_t = y_t \mid X_t = i)$$

Then the forward and backward variables satisfy simple recursions (remember that P_t is the transition matrix at time t):

$$\alpha_{t+1}(j) = \sum_{i=1}^{S} \alpha_t(i) P_t(i,j) e_{t+1}(j) \qquad \beta_t(i) = \sum_{j=1}^{S} P_t(i,j) e_{t+1}(j) \beta_{t+1}(j) \tag{4.3}$$

This can be rewritten in matrix notation:

$$\alpha_{t+1} = \operatorname{diag}(e_{t+1}) P_t^\top \alpha_t \qquad \beta_t = P_t \operatorname{diag}(e_{t+1}) \beta_{t+1} \tag{4.4}$$

The initial values are given by

$$\alpha_1 = \operatorname{diag}(e_1) p \qquad \beta_T = \mathbf{1}$$

From the forward and backward variables, one can deduce many interesting quantities. The likelihood of the observations is obtained by summing over the states in the last forward variable:

$$\mathcal{L} = \mathbb{P}(Y_{1:T} = y_{1:T}) = \sum_{i=1}^{S} \alpha_T(i) \tag{4.5}$$

The posterior state marginals are deduced by multiplying the forward and backward variables:

$$\gamma_t(i) = \mathbb{P}(X_t = i \mid Y_{1:T} = y_{1:T}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{S} \alpha_t(i)\beta_t(i)}$$

The posterior transition marginals can be computed similarly:

$$\xi_t(i,j) = \mathbb{P}(X_t = i, X_{t+1} = j \mid Y_{1:T} = y_{1:T}) = \frac{\alpha_t(i)P_t(i,j)e_{t+1}(j)\beta_{t+1}(j)}{\sum_{i=1}^{S}\sum_{j=1}^{S} \alpha_t(i)P_t(i,j)e_{t+1}(j)\beta_{t+1}(j)}$$

Note that if we are only interested in the posterior distribution of the current state X_T, then one forward pass is enough because $\gamma_T \propto \alpha_T$.

4.2.2.2 Estimation

Traditionally, estimation in HMMs is performed using the Baum-Welch algorithm (Baum et al. 1970), which is an early incarnation of the Expectation-Maximization (EM) paradigm (Dempster, Laird, and Rubin 1977). This algorithm alternates between two subroutines: the E step uses the forward-backward algorithm to compute posterior marginals γ and ξ, while the M step exploits these marginals to update parameter estimates by solving an optimization problem of the form

$$\widehat{\theta} \in \operatorname*{argmax}_{\theta} Q(\theta_{\text{old}}, \theta)$$

In standard HMMs without controls, the M step has an explicit solution for state parameters:

$$\widehat{p}(i) = \gamma_1(i) \qquad \widehat{P}(i,j) = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.6}$$

The emission parameters can also be updated easily by performing MLE. We only need to sum sufficient statistics using the state posteriors $\gamma_t(i)$ as sample weights. For instance, if we consider our example of multivariate Poisson emissions, we get:

$$\widehat{\lambda}(i,m) = \frac{\sum_{t=1}^{T} \gamma_t(i)\ \text{count}(y_t, m)}{\sum_{t=1}^{T} \gamma_t(i)\ \text{duration}(y_t)}$$

Unfortunately, as soon as we add control variables, the M step no longer has explicit solutions. A possible workaround is to apply a generalized EM algorithm. As suggested by Bengio and Frasconi (1994), one can solve the optimization problem approximately, or even replace it with a single gradient step performed on the auxiliary function $Q(\theta_{\text{old}}, \theta)$.

Instead, we follow the path of Qin, Auerbach, and Sachs (2000) and seek to directly maximize the likelihood with a gradient-based method. Our intuition is that differentiating the log-likelihood $\log \mathcal{L}(\theta)$ is cheaper than differentiating the auxiliary function $Q(\theta_{\text{old}}, \theta)$. Indeed, it only requires implementing a single forward pass instead of the full forward-backward procedure (as observed by Eisner (2016), this is more or less equivalent to letting AD derive the backward pass).

According to Equation (4.5), the likelihood is a by-product of the forward sequence (α_t). By Equation (4.3), this forward sequence can be deduced from the sequence of local transition matrices (P_t) and emission likelihoods (e_t). Both of these are obtained by combining the controls (u_t) with the parameter θ. Since all the operations we perform are amenable to AD, we can backpropagate gradients through this computational graph (see Figure 4.3) and compute $\partial(\log \mathcal{L})/\partial\theta$.

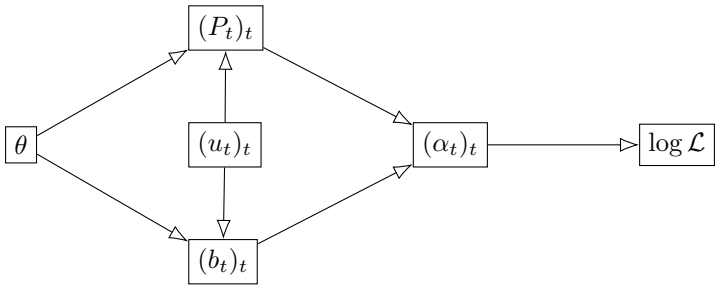


Figure 4.3: Computational graph for controlled HMM log-likelihood

4.2.2.3 Prediction

Once we have an estimate of θ, predicting the future is not overly complicated. We start by applying the forward recursion to deduce the posterior marginals γ_T of the current state X_T. Then, we compute the distribution of future states using matrix products:

$$\mathbb{P}(X_{T+h} \mid X_T \sim \gamma_T) = \gamma_T P_T P_{T+1} \cdots P_{T+h-1}$$

In turn, this allows us to deduce many interesting quantities, such as the expected hitting time of a given state.

The computational graph for prediction is very similar to that of Figure 4.3. But this time, we are more interested in computing gradients of the future state probabilities with respect to the future controls $u_{T+1}, \ldots, u_{T+h}$. This can help us adjust the behavior of the system we model: for instance, we may tune the controls to avoid or target specific states.

4.2.3 Numerical stability

When dealing with long sequences, the values of α_t and β_t tend to become very small, which leads to numerical underflow. To avoid this problem, one can rescale the forward and backward variables at each time step (Rabiner 1989). An alternative is to work directly in the logarithmic domain (Mann 2006). If we do that, then the forward-backward recursion of Equation (4.3) becomes

$$\log \alpha_{t+1}(j) = \underset{i \in [S]}{\text{logsumexp}} \left(\log \alpha_t(i) + \log P_t(i,j) \right) + \log e_{t+1}(j)$$
$$\log \beta_t(i) = \underset{j \in [S]}{\text{logsumexp}} \left(\log P_t(i,j) + \log e_{t+1}(j) + \log \beta_{t+1}(j) \right)$$

where the log-sum-exp function is defined as

$$\text{logsumexp}(v) = \log \sum_i \exp(v_i)$$

The trick is to compute this function using the following property:

$$\text{logsumexp}(v) = \max(v) + \log \sum_i \exp(v_i - \max(v))$$

```julia
# Required methods
ground_intensity(pp, t, h)
mark_distribution(pp, t, h)
ground_intensity_bound(pp, t, h)
integrated_ground_intensity(pp, h, a, b)

# Optional methods
intensity(pp, m, t, h)
log_intensity(pp, m, t, h)
```

Code sample 4.1: Interface for `AbstractPointProcess`

```julia
# Required methods
ground_intensity(pp)
mark_distribution(pp)

# Optional methods
intensity(pp, m)
log_intensity(pp, m)
```

Code sample 4.2: Interface for `AbstractPoissonProcess`

4.2.4 Example: controlled HMM with Poisson process emissions

We illustrate our discussion with a specific kind of controlled HMM, whose emissions are homogeneous Poisson processes with mark set $\mathfrak{M} = [M]$. This goes to show that emissions are not limited to finite-dimensional distributions (like a Gaussian). As long as we can draw samples and compute likelihoods, emissions can be defined by probability measures on any space.

In the standard case (without controls), the parameter θ includes the state transition matrix $P \in [0,1]^{S \times S}$, the initial state distribution $p \in [0,1]^S$ and the state-specific emission parameters $\lambda \in \mathbb{R}_+^{S \times M}$. In the controlled case, one could imagine that the transition matrix P_t and emission parameters λ_t at time t are obtained by applying neural networks to the vector of controls u_t, so that $P_t = \varphi(u_t)$ and $\lambda_t = \psi(u_t)$. Then, in addition to the initial state distribution p, the parameters θ would include the neural weights for both networks φ and ψ. Of course, adequate constraints must be enforced on the output of φ (resp. ψ) to obtain valid transition matrices P_t (resp. intensity vectors λ_t).

4.3 The package `PointProcesses.jl`

We use Poisson processes in Chapter 9 to model sequences of messages sent by condition monitoring systems. To the best of our knowledge, no other Julia package contains a generic and lightweight implementation of temporal point processes, so we decided to create one.

4.3.1 Structure

Our package `PointProcesses.jl` provides two interfaces:

- `AbstractPointProcess`, for any temporal point process (Code sample 4.1).
- `AbstractPoissonProcess`, for a homogeneous Poisson process (Code sample 4.2).

It is up to user to code the concrete subtypes they need, along with the necessary methods. Subtypes of `AbstractPointProcess` are compatible with Ogata's Algorithm 1, while subtypes of `AbstractPoissonProcess` support the more efficient Algorithm 2 for simulation. All are amenable to log-likelihood computation with Equation 4.2.

We provide two examples as part of the package: `MultivariatePoissonProcess` (with a finite mark set $\mathfrak{M} = [M]$) and `MarkedPoissonProcess` (with an arbitrary mark distribution).

```julia
using Distributions

struct MultivariatePoissonProcess{R} <: AbstractPoissonProcess{Int}
    λ::Vector{R}
end

function ground_intensity(pp::MultivariatePoissonProcess)
    return sum(pp.λ)
end

function mark_distribution(pp::MultivariatePoissonProcess)
    return Categorical(pp.λ ./ sum(pp.λ))
end
```

Code sample 4.3: Minimal source code for a multivariate Poisson process

```julia
using ForwardDiff, PointProcesses

# Construction
M = 5
pp = MultivariatePoissonProcess(rand(M))
# Simulation
tmin, tmax = 0.0, 1000.0
history = rand(rng, pp, tmin, tmax)
# Computing and differentiating the likelihood
loglikelihood(λ) = logdensityof(MultivariatePoissonProcess(λ), history)
g = ForwardDiff.gradient(loglikelihood, rand(M))
# Maximum Likelihood Estimation
pp_est = fit_mle(MultivariatePoissonProcess{Float64}, history)
```

Code sample 4.4: Example use of a multivariate Poisson process

Their implementation is very simple, as demonstrated by Code sample 4.3. Code sample 4.4 gives examples of what can be done with a `MultivariatePoissonProcess`.

In the general case, learning point process parameters relies on numerical maximization of the log-likelihood, which we leave to the user in order to keep our package lightweight. However, for our two examples of Poisson processes, direct estimation is possible. We implement it using the same formalism as the `Distributions.jl`[3] (Besançon, Papamarkou, et al. 2021; D. Lin et al. 2023), which involves explicit computation of the sufficient statistics.

4.3.2 Design choices

To ensure maximum composability with other packages, we make two important design choices. First, our approach is mostly based on interfaces, *i.e.*, abstract types with required methods. Second, all of our code is agnostic with respect to number types: as we have seen in Chapter 3, this is key to allow a wide variety of uses.

[3] https://github.com/JuliaStats/Distributions.jl

```
# Required methods
nb_states(hmm, θ)
initial_distribution(hmm, θ)
transition_matrix(hmm, θ)
emission_distribution(hmm, s, θ)
```

Code sample 4.5: Interface for `AbstractHMM`

```
# Required methods
nb_states(hmm, θ)
initial_distribution(hmm, θ)
transition_matrix(hmm, u, θ)
emission_parameters(hmm, u, θ)  # returns η
emission_distribution(hmm, s, η)
```

Code sample 4.6: Interface for `AbstractControlledHMM`

4.4 The package `ControlledHiddenMarkovModels.jl`

We use controlled HMMs in Chapter 9 to model the degradation of a train as a function of its activity. While there exist other packages for HMMs in Julia, like `HMMBase.jl`[4], none of them boasts all the features that were needed for this project, like control variables or arbitrary emission distributions. The required design overhaul to `HMMBase.jl` would have made our new code incompatible with it anyway, which is why we decided to start from scratch.

4.4.1 Structure

On paper, standard HMMs are a special case of controlled HMMs, but in practice, implementation differs significantly, which is why we separate these cases. Our package `ControlledHiddenMarkovModels.jl` thus revolves around two interfaces:

- `AbstractHMM`, for HMMs without control variables (Code sample 4.5)
- `AbstractControlledHMM`, for HMMs with control variables (Code sample 4.6)

As in `PointProcesses.jl`, we expect the user to build their own concrete subtypes. Once the necessary methods are defined, simulation, inference of the current state, log-likelihood computation and the full forward-backward procedure are made possible without further effort.

Code sample 4.7 shows how a standard HMM is implemented within our package. This can serve as a blueprint for users wishing to define more sophisticated subtypes. Meanwhile, Code sample 4.8 demonstrates the use of an `HMM` for simulation and learning. Since the syntax is a bit more cumbersome, we do not show any code for controlled HMMs, but the main ideas are very similar.

4.4.2 Design choices

Just like `PointProcesses.jl`, the design of `ControlledHiddenMarkovModels.jl` emphasizes composability by relying on abstract interfaces and working with arbitrary number types. We now list the features that make our package different from its main competitor, `HMMBase.jl`.

[4] https://github.com/maxmouchet/HMMBase.jl

```julia
struct HMM{R1,R2,D} <: AbstractHMM
    p::Vector{R1}
    P::Matrix{R2}
    emissions::Vector{D}
end

nb_states(hmm::HMM, θ=nothing) = length(hmm.p)
initial_distribution(hmm::HMM, θ=nothing) = hmm.p
transition_matrix(hmm::HMM, θ=nothing) = hmm.P
emission_distribution(hmm::HMM, s, θ=nothing) = hmm.emissions[s]
```

Code sample 4.7: Source code for a standard HMM

```julia
using ControlledHiddenMarkovModels, PointProcesses

# Construction
p = [0.3, 0.7]
P = [0.9 0.1; 0.2 0.8]
λ = [1.0 3.0; 2.0 2.0; 3.0 1.0]
emissions = [
    MultivariatePoissonProcess(λ[:, 1]),
    MultivariatePoissonProcess(λ[:, 2])
]
hmm = HMM(p, P, emissions)
# Simulation
T = 1000
state_sequence, obs_sequence = rand(rng, hmm, T)
# Learning
hmm_init = ...
hmm_est = baum_welch([obs_sequence], hmm_init; maxiter=100)
```

Code sample 4.8: Example use of a standard HMM

4.4.2.1 Compatibility with arbitrary emissions

While `HMMBase.jl` only consider emissions defined in the `Distributions.jl` package, we allow arbitrary emission objects. Our only requirement is that these objects be endowed with the `rand` and `logdensityof` methods (respectively for simulation and log-likelihood computation). This is crucial for our railway application in Chapter 9: without such flexibility, it would be impossible to create a HMM with Poisson process emissions (as seen on Code sample 4.7).

4.4.2.2 Logarithmic computations

As mentioned in Section 4.2.3, numerical stability is a major issue within the forward-backward procedure. In our implementation, densities e_t and forward-backward variables α_t, β_t are all stored normally, as we avoid underflow through repeated scaling. While performing operations in log scale is more expensive, it is also safer, especially when emissions are high-dimensional. Thus, the user needs to be given a choice.

To avoid implementing two versions of forward-backward, we leverage type genericity. The package `LogarithmicNumbers.jl`[5] allows us to work on the log scale implicitly. It represents positive numbers (such as probabilities) using their logarithm, and operations on these numbers are overloaded thanks to the logsumexp function introduced earlier. Using such numbers in our vanilla forward-backward algorithm is sufficient to avoid numerical issues. We can do that by changing the types of the HMM parameters. For instance, if we are working with Gaussian emissions that have very small variance, encoding σ as a `LogFloat64` instead of a plain `Float64` is enough to ensure that log-densities will not underflow.

4.4.2.3 Multiple sequences

Suppose we have access to several observation sequences $y^{(1)}, \dots, y^{(N)}$ with lengths $T^{(1)}, \dots, T^{(N)}$. Inference and prediction can be applied separately on each sequence. For estimation purposes, all we have to do is to sum the log-likelihoods associated with each one (for gradient-based learning), or to combine the sufficient statistics appropriately during explicit re-estimation (for the Baum-Welch algorithm).

This option is not available in `HMMBase.jl`, but we provide it as part of our Baum-Welch algorithm for standard HMMs. The only difficulty is that sufficient statistics may not exist for emissions outside the exponential family. In such cases, the user is invited to implement a method called `fit_mle_from_multiple_sequences`, which takes weighted observation sequences as input and returns an estimator for the associated emission distribution.

4.4.2.4 Internal and external parameters

As can be seen in Code samples 4.5 and 4.6, most methods accept model parameters θ as an optional argument. This may seem superfluous, given that we can also store θ in the fields of a concrete type (see Code sample 4.7). While this form of storage works well for our basic `HMM` type, it becomes less relevant for more complex models, for instance those with controls. As soon as neural networks come into play, bundling model parameters *inside* the model itself leads to mutation during training, which AD frameworks do not enjoy.

[5] https://github.com/cjdoris/LogarithmicNumbers.jl

```julia
using PointProcesses

struct PoissonHMM <: AbstractHMM end

nb_states(::PoissonHMM, θ) = length(θ.p)
initial_distribution(::PoissonHMM, θ) = make_prob_vec(θ.p)
transition_matrix(::PoissonHMM, θ) = make_trans_mat(θ.P)
emission_distribution(::PoissonHMM, s, θ) = MultivariatePoissonProcess(θ.λ[:, s])
```

Code sample 4.9: Using external HMM parameters

Instead, our formulation allows the user to decouple the model structure (architecture of φ and ψ) from the model parameters (weight values w). This functional design was inspired by the recent `Lux.jl`[6] library (Pal 2022), which is set to become a standard in large parts of the Julia ecosystem. Code sample 4.9 shows how we can mimic the Poisson HMM of Code sample 4.8 using external parameters.

4.4.2.5 Performance considerations

Since we aim for high performance, an essential caveat is the allocation of new memory (see Chapter 3). Whenever possible, we strive to minimize the memory footprint of our code by reusing the same arrays several times. For instance, when performing the forward pass to compute the likelihood, since we only need the values of $\alpha_T \in \mathbb{R}^S$, we can use a single vector instead of the full forward matrix $\alpha \in \mathbb{R}^{T \times S}$. Furthermore, the control variables u_t force us to generate a new transition matrix P_t at every time step, along with new parameters for the distribution of observations. To store them, we overwrite the arrays generated at $t = 1$ instead of allocating new ones T times.

We also take advantage of Julia's built-in matrix multiplication by using Equation (4.4) instead of (4.3) for the forward and backward passes.

4.4.2.6 Automatic differentiation

Alas, while memory reuse makes our code faster, it comes with a drawback: reverse mode AD with `Zygote.jl` does not support array mutation. To overcome this limitation, we would need to implement a custom chain rule (see Chapter 3) which backpropagates log-likelihood gradients through the recursion of Equation (4.3). For lack of time, we were not able to complete this part of the work before handing in the present manuscript, but we expect it will be done soon.

In the meantime, log-likelihoods are computed using mutation-free forward passes. Due to unresolved compatibility issues between `LogarithmicNumbers.jl` and `ForwardDiff.jl` / `Zygote.jl`, we also implement a logarithmic version of these forward passes, which is not needed as long as no AD is necessary.

[6] https://github.com/avik-pal/Lux.jl

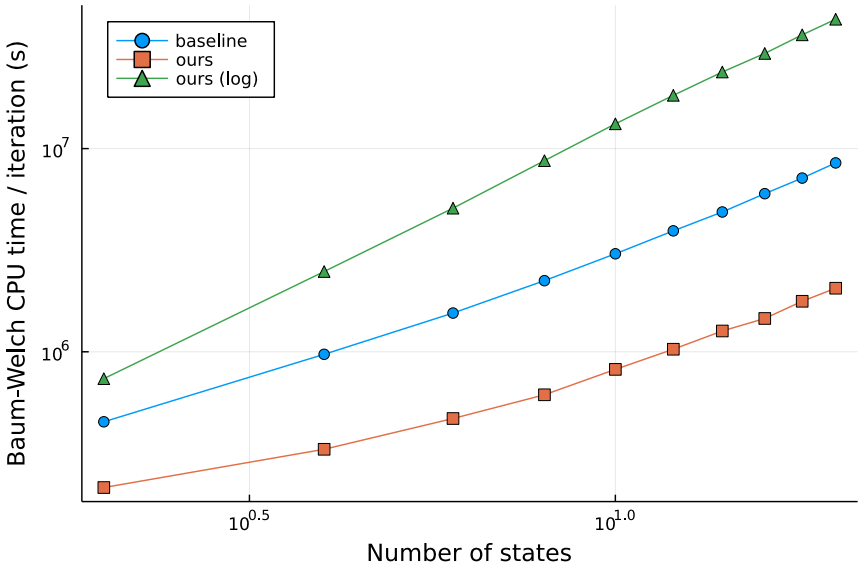


Figure 4.4: Benchmarking Baum-Welch implementations

4.5 Numerical experiments

We conclude this chapter with a performance comparison between `ControlledHiddenMarkovModels.jl` and the baseline package `HMMBase.jl`. Our test case is a standard HMM with Gaussian emissions defined by $\mu_s = s$ and $\sigma_s = 0.1$. We simulate a single observation sequence of length $T = 1000$, and define an initial model guess by $\hat{\mu}_s = s + \frac{\texttt{randn()}}{10}$ and $\hat{\sigma}_s = 1$. For both models, the transition matrix P and initial distribution p are drawn at random. The three approaches we compare are:

1. The Baum-Welch algorithm from `HMMBase.jl`, which uses logarithmic computations by default
2. The Baum-Welch algorithm from our package, with normal number types
3. The Baum-Welch algorithm from our package, with a logarithmic number type for the standard deviation $\hat{\sigma}_s$

Reassuringly, all three algorithms give rise to the exact same sequence of likelihood values, up to numerical precision. What is more interesting is to compare the estimation runtime, as is done on Figure 4.4.

As we can see, our default algorithm is significantly faster than the one from `HMMBase.jl`, mostly thanks to a better use of memory and optimized matrix multiplications. Thus, it should be used whenever numerical stability is not an issue. Indeed, as soon as we switch to the log scale, we suffer a severe performance hit and end up doing worse than the baseline. The reason is that when Julia computes `a + b` for two logarithmic numbers, under the hood it actually performs `logsumexp(loga, logb)`, which is much more expensive.

Such is the price to pay for genericity: unlike `HMMBase.jl`, our code is not *specifically designed* for logarithmic computations. In typical Julia fashion, it just *happens to work* with `Logarithmic-`

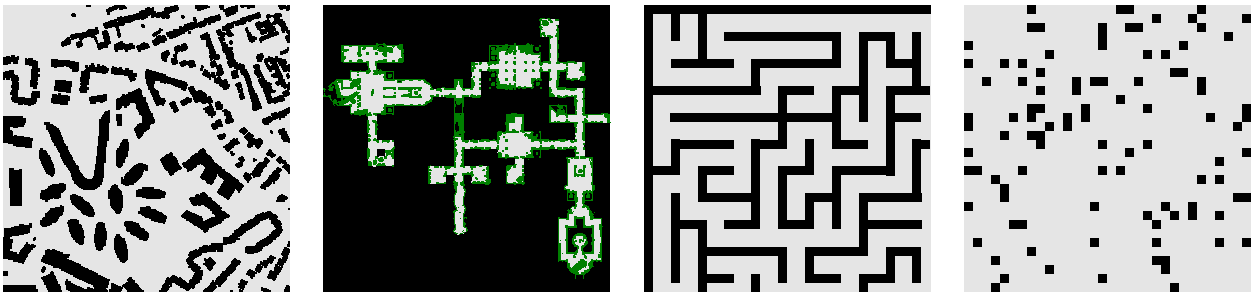
`Numbers.jl`, without even declaring the latter as a dependency. However, if we want to speed it up, a custom logarithmic forward-backward will probably be necessary.

5 Multi-Agent Path Finding

> Neo, sooner or later you're going to realize, just as I did: there's a difference between knowing the path and walking the path.
>
> *Morpheus*
> The Matrix (1999)

Contents

In this chapter, we present the Multi-Agent Path Finding (MAPF) problem, and propose a generic formulation that encompasses many application settings. We then review the main MAPF



(a) Streets of Paris (b) Video game (c) Maze (d) Random obstacles

Figure 5.1: Examples maps from the MAPF benchmark set of Stern et al. (2019)

algorithms from the literature. Finally, we introduce our open source Julia package `MultiAgentPathFinding.jl`[1], a generic toolbox for modeling and solving large scale MAPF instances. This package will be essential when we tackle railway track allocation for the Flatland challenge in Chapter 5.

5.1 Mathematical formulation of the problem

Broadly speaking, MAPF consists in planning trajectories for a set of agents on a network, such that no two agents find themselves in conflict. It has numerous concrete applications, from video games to warehouse management (see Figure 5.1). While the general principle remains the same, the specifics of the problem, for instance the definition of a conflict, vary greatly throughout the literature (Stern et al. 2019). Since we want to design a generic MAPF package, we start by proposing a flexible mathematical formulation.

5.1.1 Graph and agents

An instance of MAPF is given by a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ and a set of $A \in \mathbb{N}$ agents. Each agent $a \in [A]$ is defined by the following features: its departure vertex $v_a^{\text{dep}} \in \mathcal{V}$, its arrival vertex $v_a^{\text{arr}} \in \mathcal{V}$ and its departure time $t_a^{\text{dep}} \in \mathbb{N}$. The goal of agent a is to find a conflict-free path from v_a^{dep} to v_a^{arr}, starting at time t_a^{dep}.

By playing with the properties of the graph $\mathcal{G}$, we can authorize or forbid a wide variety of agent behaviors. For instance, waiting in place at a vertex $v \in \mathcal{V}$ is made possible by adding a self-loop (v, v) to $\mathcal{E}$. If we want the waiting to be free of cost, we simply need to set the self-loop weight $w(v, v)$ to 0. Clever graph-building also allows us to take the shape or direction of an agent into account, by adding more information to the vertices. An example is given in Chapter 11 for the orientation of a train.

Let $T \in \mathbb{N}$ be a temporal horizon. In order to keep track of agent positions through time, we need to build a time-expanded version $\mathcal{G}(T) = (\mathcal{V}(T), \mathcal{E}(T))$ of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. To clarify exposition, we will refer to elements of $\mathcal{V}(T)$ as temporal vertices, and to elements of $\mathcal{E}(T)$ as temporal edges. The set of temporal vertices $\mathcal{V}(T)$ is the Cartesian product between the time axis and the vertices:

$$\mathcal{V}(T) = \{(t, v) : t \in [T], v \in \mathcal{V}\}.$$

[1] https://github.com/gdalle/MultiAgentPathFinding.jl

The set of temporal edges $\mathcal{E}(T)$ links temporal vertices to one another by assuming that each edge requires one time step:

$$\mathcal{E}(T) = \{(t,u) \to (t+1,v) \colon t \in [T-1], (u,v) \in \mathcal{E}\}$$

Alternately, we can express temporal edges like this:

$$\mathcal{E}(T) = \{(t,e) : t \in [T-1], e \in \mathcal{E}\}$$

5.1.2 Decision variables and objective

In the time-expanded graph, the trajectory of an agent $a \in [A]$ can be represented as a path P_a going from the temporal vertex $(t_a^{\text{dep}}, v_a^{\text{dep}})$ to any temporal vertex of the form (t, v_a^{arr}). We denote by $\mathcal{P}_a$ the set of such temporal paths. A complete solution is therefore written $P = (P_a)_{a=1}^A$ and belongs to $\mathcal{P} = \bigtimes_{a=1}^A \mathcal{P}_a$. Regarding the objective, a usual choice is the *flowtime*, which is the sum of path durations. More generally, here we seek to minimize the sum of path weights:

$$\sum_{a=1}^{A} w(P_a) = \sum_{a=1}^{A} \sum_{(t,e) \in P_a} w(e)$$

5.1.3 Conflict constraints

For each vertex $v \in \mathcal{V}$, let $\mathcal{I}_v \subset \mathcal{V}$ denote the set of vertices that are incompatible with v. If v is visited by an agent a at time t, then none of the vertices in $\mathcal{I}_v$ can be visited by another agent b at the same time. Similarly, for each edge $e \in \mathcal{E}$, let $\mathcal{I}_e \subset \mathcal{E}$ denote the set of edges that are incompatible with e. If e is crossed by an agent a at time t, then none of the edges in $\mathcal{I}_e$ can be crossed by another agent b at the same time. A natural property of these sets is symmetry: if $v \in \mathcal{I}_u$, then $u \in \mathcal{I}_v$, and the same goes for edges. We now explain how to express the standard conflicts listed by Stern et al. (2019) using incompatibility sets:

- *Vertex conflict*: if $v \in \mathcal{V}$ cannot be visited by several agents at the same time, we take $\mathcal{I}_v = \{v\}$. But if the number of agents at v does not matter, then we take $\mathcal{I}_v = \emptyset$.
- *Edge conflict*: if $(u,v) \in \mathcal{E}$ cannot be crossed by several agents at the same time, we take $\mathcal{I}_{(u,v)} = \{(u,v)\}$.
- *Swapping conflict*: if the opposite edges (u,v) and (v,u) cannot be crossed at the same time, we add (v,u) to $\mathcal{I}_{(u,v)}$.
- *Following conflict*: if a vertex u cannot be visited by two different agents consecutively (with one immediately occupying the place left by the other), we add (v',u) to $\mathcal{I}_{(u,v)}$.

By default, we assume that agents appear on their departure vertex at their departure time, and that they *disappear once they reach their arrival vertex*. Another common hypothesis in the MAPF literature is "stay at arrival", but since this hypothesis seems less relevant for railway applications, we do not consider it here. In case we want more flexibility, we can replace the departure vertex of each agent with a dummy vertex, as illustrated on Figure 5.2. It serves as a waiting area and allows agents to enter the main graph later than their departure time. This trick is particularly useful to keep agents from blocking each other needlessly, and we will need it in Chapter 11 when modeling railway stations.

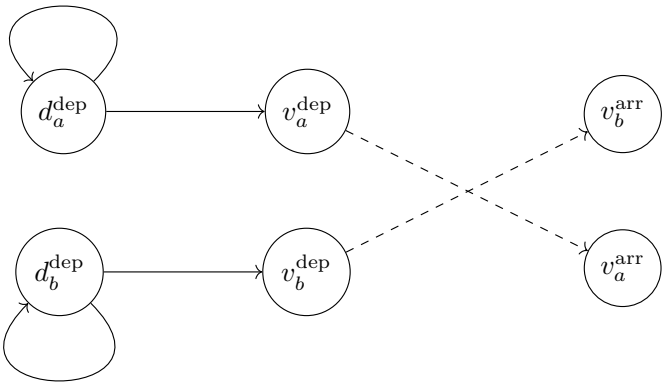


Figure 5.2: Dummy vertices as conflict-free waiting areas before departure

5.1.4 Integer Linear Program

To formulate MAPF as an Integer Linear Program (ILP), we define two sets of binary decision variables:

- $y_{a,t,e} = 1$ if and only if agent $a \in [A]$ crosses temporal edge $(t, e) \in \mathcal{E}(T)$;
- $z_{a,t,v} = 1$ if and only if agent $a \in [A]$ visits temporal vertex $(t, v) \in \mathcal{V}(T)$.

To ensure that y_a represents a valid temporal path $P_a \in \mathcal{P}_a$ in the time-expanded graph $\mathcal{G}(T)$, we use Kirchhoff flow constraints of the form

$$\sum_{u \in \delta^-(v)} y_{a,t-1,(u,v)} = \sum_{u \in \delta^+(v)} y_{a,t,(v,u)} \qquad \forall a, t, v \notin \{v_a^{\text{dep}}, v_a^{\text{arr}}\}$$

Since these constraints are linear, we can synthesize them as $F_a y_a = f_a$ using an incidence matrix F_a and inflow vector f_a. The standard MAPF problem then admits the following ILP formulation:

$$\min_{y,z} \sum_{a=1}^{A} \sum_{t=1}^{T-1} \sum_{e \in \mathcal{E}} w(e) y_{a,t,e} \quad \text{s.t.} \quad \left| \begin{array}{ll} y_{a,t,e} \in \{0,1\} & \forall a, t, e \\ z_{a,t,v} \in \{0,1\} & \forall a, t, v \\ F_a y_a = f_a & \forall a, t \\ \sum_{e \in \delta^-(v)} y_{a,t,e} = z_{a,t+1,v} & \forall a, t, v \\ \sum_{a=1}^{A} \sum_{v' \in \mathcal{I}_v} z_{a,t,v'} \leq 1 & \forall v, t \\ \sum_{a=1}^{A} \sum_{e' \in \mathcal{I}_e} y_{a,t,e'} \leq 1 & \forall e, t \end{array} \right. \tag{5.1}$$

The fourth constraint enforces coherence between vertex and edge variables. The fifth and six constraints correspond to vertex and edge conflicts respectively.

This formulation was introduced by Yu and LaValle (2013), who highlighted the similarity with multicommodity flows. Indeed, no-conflict constraints can be expressed as capacity bounds on the edges (and vertices) of the flow graph. However, tackling this ILP directly with Branch & Bound is usually not an option. That is why we now present alternative approaches.

5.2 Review of MAPF algorithms

Algorithms for solving MAPF were recently surveyed by Stern (2019). Since MAPF is an **NP**-hard problem (Surynek 2010), even finding a feasible solution is far from obvious. When it comes to large instances, the most successful algorithms rely on either decomposition or approximation. We only describe their basic principles here, and refer the reader to Felner et al. (2017) for more comments on their respective pros and cons.

5.2.1 Parallel decomposition

The first paradigm we discuss is *parallel decomposition*, in which shortest paths are computed independently and simultaneously for every agent. This corresponds to the following ILP, where the no-conflict constraints are removed, and the variable z is no longer necessary:

$$\min_{y} \sum_{a=1}^{A} \sum_{t=1}^{T-1} \sum_{e \in \mathcal{E}} w(e) y_{a,t,e} \quad \text{s.t.} \quad \left| \begin{array}{ll} y_{a,t,e} \in \{0,1\} & \forall a, t, e \\ F_a y_a = f_a & \forall a, t \end{array} \right. \tag{5.2}$$

The solution procedure can be parallelized into A separate single-agent shortest-path queries.

Of course, if we stop there, we obtain an unfeasible solution because Equation (5.2) ignores conflicts. As a consequence, various techniques have been proposed to repair conflicts *a posteriori* and output a feasible solution. They often rely on the intuition that realistic instances involve sparse interactions between agents. In their basic versions, algorithms from the parallel decomposition family return optimal solutions, at the cost of exponential complexity. However, there are ways to relax them in order to reach feasibility quicker, without pursuing provable optimality.

5.2.1.1 Independence Detection

Starting with independent shortest paths, the Independence Detection algorithm (Standley 2010) looks for a pair of agents a and b whose paths intersect. If such a collision exists, the two agents involved are merged into a meta-agent (a, b), whose position a pair of vertices that describe the positions of both its members. A new shortest meta-path is then computed for this entity, and a new conflict search is performed. This goes on until no conflicts can be found, which means the solution is both feasible and optimal. The complexity of Independence Detection is exponential in the size of the largest subset of conflicting agents.

5.2.1.2 Increasing Cost Tree Search

Sharon, Stern, Goldenberg, et al. (2011) introduce a hierarchical algorithm that divides the search space based on the length of the path for each agent. The associated Increasing Cost Tree contains nodes associated with each possible tuple of path lengths, and the root node corresponds to independent shortest paths. This tree is explored in a breadth-first manner, and node feasibility is determined using a joint shortest path query on all agents, with additional path length constraints. If the algorithm is applied without further refinements, its complexity is exponential in the number of agents.

5.2.1.3 Conflict-Based Search

The most prominent decomposition algorithm is probably Conflict-Based-Search, as suggested by Sharon, Stern, Felner, et al. (2012). If two agents a and b cross paths on vertex v at time t, they give rise to two branches of the so-called Constraint Tree: one which forbids (v, t) for agent a, and the other which forbids it for agent b. The optimal solution has to be on one of these branches, and so independent shortest paths are recomputed on each side with these additional constraints. The tree search goes on in a best-first order until a conflict-free solution is discovered. Remarkably, this algorithm only requires (constrained) single-agent shortest path queries. Numerous heuristics and enhancements have been incorporated into Conflict-Based Search over the years. See Boyarski, Felner, Stern, et al. (2015) for an early example and Boyarski, Felner, Bodic, et al. (2021) for some of the latest additions.

5.2.2 Sequential approximation

The second paradigm we discuss is *sequential approximation*, in which agents plan their paths one after the other, instead of all at once. The seminal algorithm in this family is *Cooperative A** (Silver 2005), often referred to in the literature as Prioritized Planning. Its central ingredient is a priority ordering on the set of agents. When its turn comes to seek a path, each agent takes care to avoid temporal vertices or edges that are already known to be occupied by higher-ranked agents. This is done with the help of a reservation table, which is filled by each agent after its path is chosen.

5.2.2.1 A suboptimal but fast algorithm

By treating agents sequentially and not jointly, Cooperative A* enjoys a polynomial complexity: it only requires A individual shortest path queries with reservation constraints. Of course, this method is not guaranteed to return an optimal solution. In some degenerate settings, it may even fail to find an obvious feasible solution. However, if a solution is found, it is guaranteed to be conflict-free.

Recently, Cooperative A* has been applied in conjunction with Large Neighborhood Search (LNS). To improve a feasible solution, Li, Z. Chen, Harabor, et al. (2021) propose iteratively destroying and reconstructing paths for small subsets of agents: they call this algorithm MAPF-LNS. To repair an infeasible solution, Li, Z. Chen, Harabor, et al. (2022) modify MAPF-LNS by minimizing the number of conflicts during path reconstruction, instead of banning conflicts altogether: they call this algorithm MAPF-LNS2. Search routines like these compensate the greedy aspect of Cooperative A*, and they have proven competitive on large instances, such as those of the Flatland challenge (see Chapter 11).

5.2.2.2 Lexicographic linear formulation

Unlike the independent and continuous approximations mentioned earlier, the sequential approach is not easily formulated as an ILP. However, assuming that agents are ordered from 1 to A by

decreasing priority, we can express it as a sequence of ILPs:

$$\text{for } a = 1 \text{ to } A, \quad \min_{y_a, z_a} \sum_{t=1}^{T-1} \sum_{e \in \mathcal{E}} w(e) y_{a,t,e} \quad \text{s.t.} \quad \left| \begin{array}{ll} y_{a,t,e} \in \{0,1\} & \forall t, e \\ z_{a,t,v} \in \{0,1\} & \forall t, v \\ F_a y_a = f_a & \forall t \\ \sum_{e \in \delta^-(v)} y_{a,t,e} = z_{a,t+1,v} & \forall t, v \\ \sum_{b=1}^{a} \sum_{v' \in \mathcal{I}_v} z_{b,t,v'} \leq 1 & \forall v, t \\ \sum_{b=1}^{a} \sum_{e' \in \mathcal{I}_e} y_{b,t,e'} \leq 1 & \forall e, t \end{array} \right. \tag{5.3}$$

In order words, we find the optimal path P_1 for the first agent, then fix it, move on to the second agent, find an optimal path P_2 that does not intersect with P_1, fix this one, *etc.* If instead we only fix the *objective* of higher-priority agents, but allow their path to change (without getting longer) in the following iterations, we obtain the following sequence of problems:

$$\text{for } a = 1 \text{ to } A, \quad \min_{y_{1:a}, z_{1:a}} \sum_{t=1}^{T-1} \sum_{e \in \mathcal{E}} w(e) y_{a,t,e} \quad \text{s.t.} \quad \left| \begin{array}{ll} y_{b,t,e} \in \{0,1\} & \forall b \in [a], t, e \\ z_{b,t,v} \in \{0,1\} & \forall b \in [a], v \\ F_b y_b = f_b & \forall b \in [a], t \\ \sum_{e \in \delta^-(v)} y_{b,t,e} = z_{b,t+1,v} & \forall b \in [a], t, v \\ \sum_{b=1}^{a} \sum_{v' \in \mathcal{I}_v} z_{b,t,v'} \leq 1 & \forall v, t \\ \sum_{b=1}^{a} \sum_{e' \in \mathcal{I}_e} y_{b,t,e'} \leq 1 & \forall e, t \end{array} \right. \tag{5.4}$$

The nuance is subtle, but in this second case, we end up with a lexicographic ILP, something we revisit in Chapter 8.

5.2.3 Single-agent problem

Most of the algorithms listed above assume that we are able to construct the optimal path for a single agent very efficiently. This can be done using a variety of well-known shortest path algorithms. On large graphs, the most efficient one is often A* (Hart, Nilsson, and Raphael 1968), because it exploits custom heuristics to quickly discard hopeless partial paths during enumeration. On very large graphs such as road networks, A* may still be too slow for practical purposes. Bast et al. (2016) discuss how to make it faster by leveraging advanced heuristics. In particular, if we must solve a sequence of problems on the same network, preprocessing the graph structure is a good way to improve upon naive lower bounds.

When used to provide solutions for MAPF, the A* algorithm must sometimes respect reservation constraints of the form "do not visit vertex v at time t". This means A* needs to run on a time-expanded graph, with one vertex per couple (t, v). In this scenario, an admissible heuristic is given by the length of the shortest path to the destination, computed without reservation constraints. Note that the time-expanded graph should never be stored explicitly: instead, we can generate its nodes and arcs on the fly during temporal loops.

5.3 The package `MultiAgentPathFinding.jl`

We conclude this chapter by describing our Julia package `MultiAgentPathFinding.jl`, which we reuse in Chapter 11. The code we present is often modified or shortened for clarity: actual source code can be found on the GitHub repository.

5.3.1 Problem and solution storage

Our package is built around three main structures:

- `MAPF` describes a problem instance as a tuple $(\mathcal{G}, w, \mathcal{I}, v^{\text{dep}}, v^{\text{arr}}, t^{\text{dep}})$ (Code sample 5.1).
- `TimedPath` stores a path through the temporal graph (Code sample 5.2), which means a full `Solution` is a vector of `TimedPath`.
- `Reservation` contains sets of occupied temporal vertices / edges (Code sample 5.3).

Along with these structures, various utilities are provided to identify conflicts, check solution feasibility and compute objective functions.

```
struct MAPF
    g  # graph G
    departures  # vector a -> v_a^dep
    arrivals  # vector a -> v_a^arr
    departure_times  # vector a -> t_a^dep
    vertex_conflicts  # dict v -> I_v
    edge_conflicts  # dict (u,v) -> I_uv
    edge_weights_vec  # vector w
    flexible_departure  # true / false
end
```

Code sample 5.1: Instance storage

```
struct TimedPath
    tdep  # departure time
    path  # vector of vertices
end
```

Code sample 5.2: Temporal path storage

```
struct Reservation
    forbidden_vertices  # {(t, v)}
    forbidden_edges  # {(t, u, v)}
end
```

Code sample 5.3: Reservation storage

To make the implementation generic, we only expect the graph object $\mathcal{G}$ to follow the `Graphs.jl`[2] interface (Fairbanks et al. 2021), which is the Julia standard in this field. For instance, Code sample 5.4 shows how to quickly create an instance of MAPF on a grid graph. When left unspecified, departure times are all set to 1, and conflict sets are given default values which only consider vertex, edge and swapping conflicts (no following conflicts). The `flexible_departure` field specifies whether departure can happen after t^{dep} (it has a similar effect as the graph gadget of Figure 5.2).

```
using MultiAgentPathFinding

L, A = 10, 20
g = Graphs.grid([L, L])
departures = rand(1:L^2, A)
arrivals = rand(1:L^2, A)
mapf = MAPF(g, departures, arrivals; flexible_departure=true)
```

Code sample 5.4: Constructing a MAPF instance from a grid graph

[2] https://github.com/JuliaGraphs/Graphs.jl

5.3.2 Solution algorithms

At the moment, `MultiAgentPathFinding.jl` focuses on sequential approximation algorithms, since they scale effortlessly to large instances. Note that another package with the same name[3] exists, with an implementation of Conflict-Based Search. However, it is left unmaintained and was never intended to become a general-purpose library.

5.3.2.1 Single-agent problem

Because we need to deal with reservation constraints, we cannot reuse the default A* provided in `Graphs.jl`. We thus need to reimplement it, and we use this opportunity to propose some improvements. Dijkstra's algorithm and A* both work with a priority queue: whenever a vertex u is visited, each neighbor v is inserted in the queue with an adequate priority value. As highlighted by M. Chen et al. (2007), insertion can proceed in one of two ways. Option 1 is to check the existence of v in the queue before insertion, and update its priority if the queue already contains it (Algorithm 3) Option 2 is to insert v with a new priority every time (Algorithm 4).

Algorithm 3: Dijkstra with priority updates

```
Q ← ∅
D[v] ← +∞ for all v
insert(Q, s, 0)
while Q ≠ ∅ do
    (u, d) ← extract_min(Q)
    D[u] ← d
    foreach v ∈ 𝒩(u) do
        if D[u] + w[u, v] ≤ D[v] then
            if D[v] = +∞ then
                insert(Q, v, D[u] + w[u, v])
            else
                update(Q, v, D[u] + w[u, v])
            end
            D[v] ← D[u] + w[u, v]
        end
    end
end
```

The first option is the one chosen by `Graphs.jl`, with a `PriorityQueue` taken from the package `DataStructures.jl`[4]. The second option is the one we choose for `MultiAgentPathFinding.jl`. It makes insertion less costly and allows us to work with simpler data structures. For instance, the `PriorityQueue` can be replaced with a faster `BinaryHeap`. However, it also leads to more elements in the queue, which means the resulting trade-off must be evaluated carefully.

To justify our choice, we benchmark Algorithm 3 (with a `PriorityQueue`) and Algorithm 4 (with a `BinaryHeap`) against the version of Dijkstra available in `Graphs.jl`. The task is finding the shortest path from the top left to the bottom right corner of a grid with random weights. The

[3] https://github.com/Shushman/MultiAgentPathFinding.jl
[4] https://github.com/JuliaCollections/DataStructures.jl

Algorithm 4: Dijkstra without priority updates

```
Q ← ∅
D[v] ← +∞ for all v
insert(Q, s, 0)
while Q ≠ ∅ do
    (u, d) ← extract_min(Q)
    if d < D[u] then
        D[u] ← d
        foreach v ∈ 𝒩(u) do
            if D[u] + w[u, v] ≤ D[v] then
                insert(Q, v, D[u] + w[u, v])
                D[v] ← D[u] + w[u, v]
            end
        end
    end
end
```

results are displayed on Figure 5.3. While the version with priority updates behaves very much like the `Graphs.jl` reference, we see that giving up on priority updates improves both CPU time and memory use by a significant margin. Therefore, we adopt this method to store vertices in our implementations of Dijkstra's algorithm and A*.

5.3.2.2 Cooperative A*

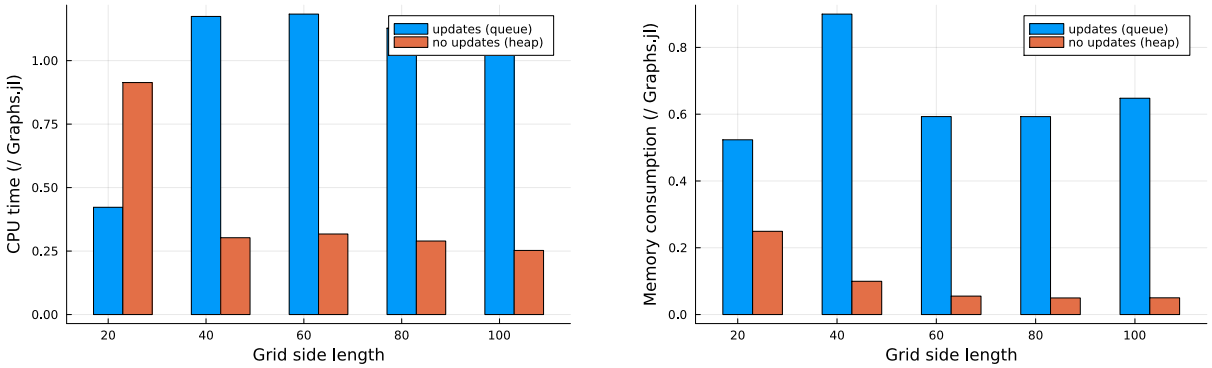
Our take on Cooperative A* follows the original paper of Silver (2005) quite closely. As discussed, we rely on a temporal A* with reservation constraints and a custom `BinaryHeap` storage for the queue. Heuristic values for all distinct arrival vertices are computed beforehand by running Dijkstra's algorithm in from all arrival vertices. This choice is justified by the application to railways in Chapter 11: since departures and arrivals are restricted to railway stations, there are very few distinct arrival vertices. As an alternative, we could replace reverse Dijkstra with reverse resumable A*, an idea that is described in Silver (2005).

While Cooperative A* is fast, it often fails to reach a feasible solution. This can sometimes be improved thanks to a better permutation of agents. Ma et al. (2019) suggest searching for the best possible permutation, while Li, Z. Chen, Harabor, et al. (2021) run Cooperative A* repeatedly with random priority orders. We do not implement these refinements in our package, choosing to rely on local search instead. However, the user is invited to input a permutation of agents, with $[\![1, A]\!]$ being the default choice.

5.3.2.3 Optimality search

The greediness of Cooperative A* implies that its output still has a lot of room for improvement. That is why we implement the MAPF-LNS algorithm of Li, Z. Chen, Harabor, et al. (2021), which we rename to *optimality search*. This algorithm starts from a feasible solution P and iteratively performs the following operations:

1. Select a subset $S \subset [A]$ of agents.



(a) CPU time

(b) Memory consumption

Figure 5.3: Comparison of priority structures for Dijkstra's algorithm on a grid graph

2. Remove their current paths $P_S^- = (P_a^-)_{a \in S}$.
3. Plan new conflict-free paths $P_S^+ = (P_a^+)_{a \in S}$.
4. Compare the objective values of solutions P and $(P \backslash P_S^-) \cup P_S^+$.
5. Keep the better of the two.

Each search iteration explores a large neighborhood, because there are many ways to complete the partial solution $P \backslash P_S^-$. Instead of naive enumeration, an arbitrary MAPF algorithm can be used to solve this subproblem. Due to its speed, Cooperative A* is the replanning routine recommended by Li, Z. Chen, Harabor, et al. (2021). There are also many ways to choose the agent subset S. Our package only provides a random neighborhood selection, which was shown by Li, Z. Chen, Harabor, et al. (2021) to be "surprisingly effective" for congested instances.

5.3.2.4 Feasibility search

In addition to optimality search, we also implement a variant of the MAPF-LNS2 algorithm of Li, Z. Chen, Harabor, et al. (2022), which we rename to *feasibility search*. This algorithm starts from an infeasible solution (with conflicts between agents), and tries to improve it iteratively. Its basic mechanism is the same as for optimality search: select some agents, remove and replan their paths, keep the new solution if progress has been made. However, there is a key difference in the replanning step: optimality search treats conflicts as a hard constraint, while feasibility search only penalizes them.

- Optimality search looks for new paths P_S^+ which have no conflict with each other or with $P \backslash P_S^-$, and whose weight is minimal.
- Feasibility search looks for new paths P_S^- whose number of conflicts with each other and with $P \backslash P_S^-$ is minimal, breaking ties with the weight.

When tackling this multiobjective replanning subproblem, Li, Z. Chen, Harabor, et al. (2022) observe that temporal A* may be very inefficient. If the optimal new path has 1 conflict, then every path

with 0 conflict must be explored and discarded before the optimum can be found. In case there is no upper bound on the time horizon, the search may never even terminate. To overcome this difficulty, the original MAPF-LNS2 replaces temporal A* with Safe Interval Path Planning (Phillips and Likhachev 2011).

However, this algorithm is rather advanced and requires careful bookkeeping, which is why we propose a simpler alternative. Instead of minimizing the number of conflicts and then the path weight, we minimize a weighted sum of both terms. We start by assigning a very low cost to the conflict term, in order to boost exploration. Then, as the LNS iterations go by, we geometrically increase the conflict cost to direct the search towards feasibility. This procedure, which is inspired by simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983), allows us to use temporal A* without giving up on termination. It is the one we implement in our package.

5.3.2.5 Double search

Once we have feasibility search and optimality search at our disposal, it is tempting to combine them. This is what our *double search* algorithm does: instead of initializing optimality search with a run of Cooperative A*, it runs a feasibility search. As we will see below, this tweak performs badly in some situations, but very well on others.

5.4 Numerical experiments

5.4.1 Benchmark instances and setting

We showcase the capabilities of `MultiAgentPathFinding.jl` on the set of MAPF benchmark instances described by Stern et al. (2019). These grid maps are part of a larger suite introduced by Sturtevant (2012), and they are all freely available on his website[5]. Some examples are given on Figure 5.1. To convert instances into `MAPF` objects, we use two small packages of our own making:

- `GridGraphs.jl`[6] represents graphs as grids of vertices. It can accommodate various move structures (*e.g.*, move like a rook or a queen on a chessboard), as well as missing cells in the grid.
- `MAPFBenchmarks.jl`[7] reads benchmark files and parses the appropriate `GridGraph` along with the set of agents.

We use a setting similar to the one laid out by Stern et al. (2019). Vertex, edge and swapping conflicts are forbidden, while following conflicts are permitted. Grid cells are linked with their top, bottom, left and right neighbors (no diagonal moves), and with themselves for waiting purposes. All agents depart at the same time, and the objective is to minimize the flowtime.

The main difference between our benchmark and theirs is the behavior of agents at departure and arrival. First, we reject the "stay at arrival" assumption, instead allowing agents to disappear after reaching their arrival vertex. Second, we set `flexible_departure` to `true`, thus making it possible for agents to depart at any time after t^{dep}. These two choices combined have a very interesting consequence: no matter the ordering, Cooperative A* will *always* find a feasible solution.

[5] https://movingai.com/benchmarks/mapf.html
[6] https://github.com/gdalle/GridGraphs.jl
[7] https://github.com/gdalle/MAPFBenchmarks.jl

Parameter description	Value
Max duration for optimality search (seconds)	5
Max duration for feasibility search (seconds)	5
Number of agents selected in a neighborhood	10
Initial cost of a conflict in feasibility search	1.0
Multiplicative rate of conflict price increase	0.1

Table 5.1: Algorithm parameters for our MAPF benchmark

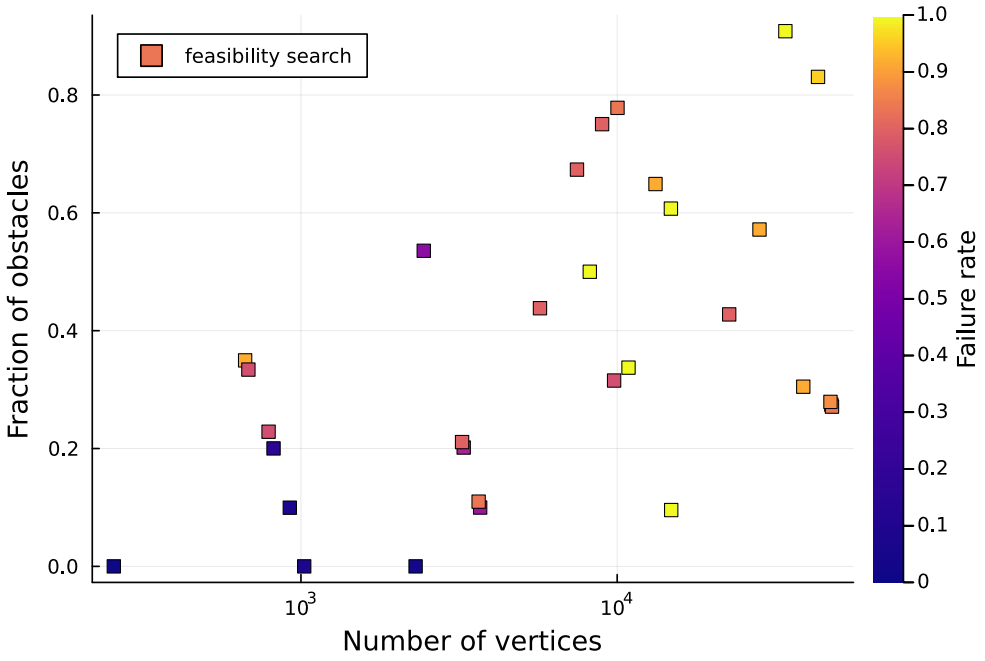


Figure 5.4: Success rate of feasibility search on the MAPF benchmark set

Our goal is to demonstrate and compare four algorithms, whose parameters are summed up in Table 5.1: cooperative A*, optimality search, feasibility search and double search. Each instance comes with 25 scenarios of 1000 agents, but we only use the first 100 agents of each scenario.

5.4.2 Results

We were able to run our code on every instance except the largest one (`orz900d.map`), which caused an out of memory error. The cause of this error will be investigated in the near future, it might have something to do with our use of reverse Dijkstra on the full graph for heuristic computation.

Our main performance measure is the *optimality gap*, which measures the relative excess cost with respect to an optimal solution. Since we do not have access to optimal solutions, we upper-bound this gap using the flowtime of independent shortest paths as a (loose) lower bound on the optimal value. We are also interested in the *success rate*, that is, the percentage of scenarios (within a given instance) where a feasible solution is found. Results are presented on Figures 5.4 and 5.5.

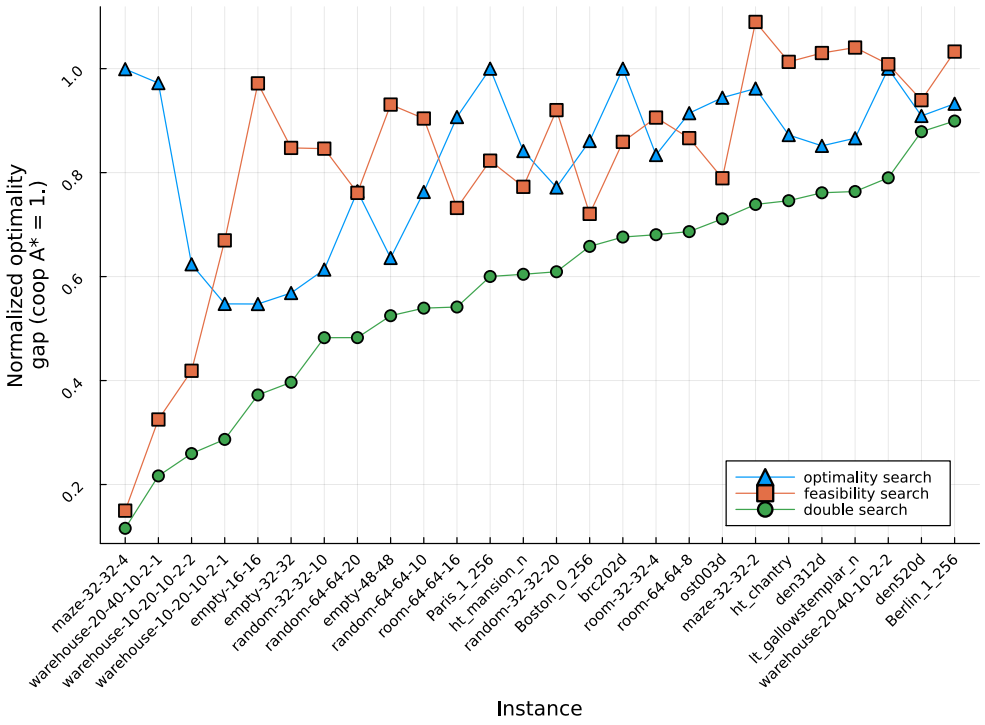


Figure 5.5: Optimality gaps of 3 algorithms on a MAPF benchmark subset

Figure 5.4 displays the success rate of the feasibility search as a function of instance features. The number of vertices on the grid is a proxy for instance difficulty, just like the fraction of obstacles. Without much surprise, we observe that larger and more labyrinth-like instances make it harder to find a feasible solution. Of course, in some cases, this could be solved by simply increasing the maximum duration of feasibility search.

Figure 5.5 ranks all algorithms based on their optimality gap, with Cooperative A* as a baseline. To enable fair comparison, it only shows the subset of scenarios on which feasibility search reaches a feasible solution. Depending on the instance, we see that optimality search and feasibility search each have their own strengths, which is why combining them usually makes a lot of sense.

Part II

Theoretical contributions

6

Minimax estimation of partially-observed vector autoregressions

It's not who I am underneath. It's what I do that defines me.

Bruce Wayne
Batman Begins (2005)

Contents

This chapter corresponds to our preprint D. and de Castro (2022).

6.1 Introduction

Time series provide a natural representation for periodic measurements of a stochastic process. In particular, those defined by linear Gaussian recursions may be the most widely used and the easiest to study. Well-known examples include the AutoRegressive (AR) process and its multivariate counterpart, the Vector AutoRegressive (VAR) process.

Industrial applications of these models encounter two main challenges. First, they often involve signals in high dimension, which means sparsity assumptions play an important role. Second, the variables of interest are rarely measured exactly or entirely. Indeed, physical constraints such as the cost of sensors can make it impossible to capture every component of the system's state at all times. It is therefore natural to ask: *how much harder does high-dimensional learning become when one only observes a fraction of the relevant values?*

6.1.1 A theoretical question

To answer this question, we study a state-space model where the state $X_t \in \mathbb{R}^D$ follows a VAR process of order 1 over a period of length T. Since the dimension D of X_t is high, we assume that its transition matrix $\theta \in \mathbb{R}^{D\times D}$ is s-sparse (there are no more than s non-zero coefficients in each row). However, we do not observe the state itself: our observations Y_t only involve the subset of components $X_{t,d}$ for which $\pi_{t,d} = 1$, where π_t is a vector of Bernoulli variables. To make matters worse, this subset is corrupted with noise, which leads to the following generative procedure:

$$X_t = \theta X_{t-1} + \mathcal{N}(0, \sigma^2 I) \qquad \pi_{t,d} \sim \mathcal{B}(p) \qquad Y_t = \operatorname{diag}(\pi_t) X_t + \mathcal{N}(0, \omega^2 I). \tag{6.1}$$

When we write $\pi_{t,d} \sim \mathcal{B}(p)$, we mean that the marginals of the sampling variables are identical, which requires that every state component be sampled with equal probability p. However, we reject the standard independence assumption in favor of temporal dependencies between the Bernoulli variables $\pi_{t,d}$ (see Section 6.1.2 for a practical justification).

To shed light on the properties of our model, we start by constructing a sparse estimator for θ, whose non-asymptotic error we upper-bound. We complement this finding with a lower bound on the minimax error that is independent of the choice of estimator. Upper and lower bound match in most regards, which proves their optimality. A rough summary of our analysis is that the best possible estimator $\widehat{\theta}$ satisfies

$$\|\widehat{\theta} - \theta\|_\infty \lesssim \left(1 + \frac{\omega^2}{\sigma^2}\right) \frac{s\sqrt{\log D}}{p\sqrt{T}} \tag{6.2}$$

with high probability. We observe that the error only depends logarithmically on the state dimension D: the major role is played by the sparsity s of the transition matrix. As expected, it decreases linearly as p grows, since more information becomes available. Lastly, it is a function of ω^2/σ^2, which means that precise recovery of θ is only possible when the noise is not too much larger than the signal.

Novel features of our work include the first proof of a minimax lower bound in this setting (to the best of our knowledge), the investigation of temporal correlations within the sampling process, the combination of discrete and continuous concentration inequalities to obtain error estimates, as well as detailed numerical experiments on simulated data.

6.1.2 A concrete application

Our study was inspired by concrete questions related to delay propagation on railway networks, which came up during a collaboration with a leading railway company. When external factors (weather, passenger behavior, mechanical failures) trigger a primary delay, resource conflicts between trains can amplify the initial incident and send ripple effects through the whole network. Understanding and predicting this propagation phenomenon is a crucial task for traffic management and robust scheduling.

To model it, we construct a network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ linking the railway stations, and we assume the existence of a hidden congestion variable $X_{t,d}$ that lives on the edges $d \in \mathcal{E}$. This congestion evolves according to a VAR process, whose transition matrix θ represents pairwise interactions between edges. The sparsity structure of θ expresses the local nature of delay propagation, which is why it is closely related to the adjacency structure of $\mathcal{G}$. Indeed, between times t and $t+1$, edges are expected to transmit congestion to their close neighbors, and not to regions of the network that are very far away.

Unfortunately for us, X_t is never observed directly. The only information we have is collected by the trains whenever they cross an edge of the network. The crossing time of a train is influenced by the congestion, but also by other individual factors: in this sense, our observations Y_t are a noisy version of the underlying process X_t. Furthermore, the observations are limited in size: the dimension of X_t is the number of edges $D = |\mathcal{E}|$, while the dimension of Y_t is linked to the number of trains on the timetable and the length of their respective journeys. We can thus define a random variable $\pi_{t,d}$ equal to 1 if a train crosses edge d between t and $t+1$, and 0 otherwise. A more realistic model would account for the possibility of multiple trains crossing an edge in the same time step, especially if the discretization interval is large. However, our binary assumption greatly simplifies exposition without betraying the qualitative behavior of the system. Crucially, this sampling mechanism exhibits temporal correlations: periods of dense traffic are likely to be followed by dense traffic, which means that the sequence of sampling variables $\pi_{t,d}$ is not independently distributed.

We recognize the framework of Equation (6.1), and can therefore apply the theoretical result of Equation (6.2). Such an error quantification provides useful insight on the estimation of θ, which

is essential to help railway operatives dimension their data sets or evaluate prediction uncertainty. This model will be revisited in Chapter 10.

6.1.3 Outline

Section 6.2 is dedicated to a literature review on VAR processes with partial observations. In Section 6.3, we define the generative procedure behind the partially-observed VAR process, and we present a sparse estimator of the transition matrix. We then state both of our theoretical results in Section 6.4: an upper-bound on the error of our specific estimator, complemented by a minimax lower bound on the error of any estimation algorithm. Section 6.5 contains numerical experiments demonstrating the impact of various parameters.

Appendix 6.A is dedicated to proving the convergence rate of the sparse estimator, while Appendix 6.B contains the derivation of the minimax lower bound.

6.2 Related work

The theory of VAR processes has been known for a long time: the book of Lütkepohl (2005) provides a detailed account. If we have full and noiseless observations of the process X_t, we can use conditional Least Squares to estimate θ by minimizing the quadratic error $\sum_t \|X_t - \theta X_{t-1}\|_2^2$. This is equivalent to solving the Yule-Walker equation $\Gamma_h = \theta \Gamma_{h-1}$, where we replace the autocovariance matrix $\Gamma_h = \mathrm{Cov}[X_{t+h}, X_t]$ with its empirical counterpart $\widehat{\Gamma}_h$. In the case of Gaussian innovations, both approaches coincide with the Maximum Likelihood Estimator (MLE).

Neither of these methods was initially designed for missing or noisy data. Luckily, statistical estimation with imprecise measurements has been thoroughly studied (Buonaccorsi 2010). The same goes for incomplete data sets ; an extensive survey was recently published by Little and Rubin (2020). According to their terminology, our work deals with data that is missing completely at random (MCAR), which means that the projection π_t is independent of the underlying process X_t. We also assume to know the distribution of the missingness indicators $1 - \pi_{t,d}$, which is not necessarily true for other applications (e.g. clinical trials).

A principled approach to deal with missing data would require extending the MLE to partially-observed time series, also known as state-space models (Cappé, Moulines, and Rydén 2005). Most of the time, exact or approximate inference is achievable using some version of the Kalman filter (Kalman 1960) or particle methods (Doucet, Godsill, and Andrieu 2000), whereas parameter estimation typically involves the Expectation-Maximization (EM) algorithm (Shumway and D. S. Stoffer 1982). Unfortunately, the EM algorithm is hard to analyze explicitly in terms of statistical error, which is why other methods are sometimes preferred in theoretical studies. In particular, plug-in methods that use covariance estimates within the Yule-Walker equation have been quite popular in the ML community.

In this line of work, the core challenge is the high dimension D of the VAR process X_t. To address it, many authors use sparsity-inducing penalties as a way to reduce data requirements and computational workload. In the last ten years, the LASSO (Tibshirani 1996) has been increasingly applied to random designs exhibiting correlations or missing data. This trend started with the seminal work of Loh and Wainwright (2012), and numerous other papers followed (see for example Basu and Michailidis 2015; Kock and Callot 2015; Melnyk and Banerjee 2016; Jalali and Willett 2018).

As an alternative to the LASSO, the Dantzig selector (Candes and Tao 2007) enforces sparsity in the objective and data fidelity in the constraints. While the LASSO requires solving a Quadratic Program (QP), for instance with proximal methods, the Dantzig selector gives rise to a Linear Program (LP) which can be parallelized across dimensions. Han, Lu, and Liu (2015) studied its application to VAR estimation, obtaining finite-sample error bounds with very natural hypotheses. A little later, Rao et al. (2017a) extended these results to the more general scenario in which a hidden VAR process is randomly sampled or projected, and then corrupted with noise. This last work is quite similar to ours, but we think that the proof they present to control the non-asymptotic error is incomplete at best[1].

Another salient feature of our paper is the search for a minimax lower bound, which allows us to prove the optimality of our convergence rates. To the best of our knowledge, this was only attempted once for partially-observed VAR processes. Rao et al. (2017b) presented a lower bound on the minimax error in a setting very similar to ours, but their result is less generic in several regards. Indeed, we account for the possibility of temporal correlations within sampling, as well as observation noise. Moreover, unlike the one proposed by Rao et al. (2017b), our proof focuses on geometric properties and does not make use of the admissible set of transition matrices until the very end. This makes it easy to handle many types of structured transitions without additional work: sparse, Toeplitz, banded, *etc.*

Finally, the error bounds we obtain are backed up by detailed numerical experiments on simulated data, which allow us to visualize the influence of every parameter of interest.

6.3 The partially-observed VAR process and its sparse estimator

Before stating our theoretical results, we introduce our statistical model and the estimator we use.

6.3.1 Model definition

The model we study was described approximately in the introduction. We now fill in the gaps of the generative procedure it relies on. The underlying state $X = (X_t)_{t\in[T]}$ follows a stationary VAR process of order 1. This process has dimension D and the following recursive definition:

$$X_t = \theta X_{t-1} + \varepsilon_t \qquad \text{with} \qquad \varepsilon_t \sim \mathcal{N}(0, \Sigma). \tag{6.3}$$

Here $\theta \in \mathbb{R}^{D\times D}$ is the transition matrix and $\Sigma \in \mathbb{R}^{D\times D}$ is the covariance matrix of the innovations (in the introduction, we assumed $\Sigma = \sigma^2 I$). To ensure stationarity of the VAR process, we must constrain the spectral radius of θ to satisfy $\rho(\theta) < 1$. Throughout the paper, we actually make the following (slightly stronger) assumption on the spectral norm of θ: there exists $\vartheta \in (0, 1)$ such that for all the values of θ we consider, $\|\theta\|_2 \leq \vartheta < 1$. Furthermore, we only study row-sparse transition matrices, having at most s nonzero coefficients in each row. In other words, we restrict our choice of parameters to

$$\theta \in \Theta_s \quad \text{where} \quad \Theta_s = \{\theta \in \mathbb{R}^{D\times D} : \|\theta\|_2 \leq \vartheta < 1 \quad \text{and} \quad \forall i, \|\theta_{i,\cdot}\|_0 \leq s\}. \tag{6.4}$$

[1]Indeed, the combination of discrete and Gaussian concentration inequalities as performed on page 2 (middle of right column) of the supplementary material for Rao et al. (2017a) glosses over the fact that L_F is itself a random variable. As we will discover during our own proof, this introduces an additional difficulty and forces us to use a more complex Gaussian concentration result (Lemma 6.A.6). See https://web.stanford.edu/~milind/papers/system_id_icassp_proof.pdf for the supplementary material in question.

We denote by $\sigma^2_{\min} = \lambda_{\min}(\Sigma)$ and $\sigma^2_{\max} = \lambda_{\max}(\Sigma)$ the minimum and maximum eigenvalues of the covariance matrix Σ.

The observation mechanism we chose implies that we do not have direct access to the latent process X_t. To construct the observations Y_t, we sample a subset of state components according to the binary vectors π_t. Then, independent Gaussian noise with variance ω^2 is added to these selected components, and we observe the result. If we denote by $\Pi_t = \operatorname{diag}(\pi_t)$ the diagonal projection matrix, we have

$$Y_t = \Pi_t X_t + \eta_t \qquad \text{with} \qquad \eta_t \sim \mathcal{N}(0, \omega^2 I). \tag{6.5}$$

An essential hypothesis we make is the mutual independence between our three sources of randomness: the innovations ε_t, the projections π_t and the observation noise η_t.

A major feature of the present work is the non-deterministic selection of observed state components, that is, the fact that π_t is a random sequence of Bernoulli vectors following a known distribution. In order to sum up the amount of information available using one parameter $p \in (0, 1)$, we want this distribution to satisfy the following condition: each component $X_{t,d}$ of the latent state must be sampled with the same marginal probability $p = \mathbb{P}(\pi_{t,d} = 1)$.

On the other hand, we also want to introduce temporal dependencies between the projections. The simplest way to achieve that is through a Markovian hypothesis: independently along each dimension d, time indices t are selected for observation according to a binary-valued Markov chain with transition matrix $P = \left(\begin{smallmatrix} 1-a & a \\ b & 1-b \end{smallmatrix}\right)$. Its coefficients are chosen to make the chain stationary with invariant measure $(\frac{b}{a+b}, \frac{a}{a+b}) = (1-p, p)$. Note that when $a = 1 - b = p$, this reduces to independent sampling of each component with probability p. We also assume there exists a universal constant χ such that $0 < \chi \leq a, b \leq 1 - \chi < 1$: this means that the chain does not transition too fast nor too slowly.

Our data set is built from N independent realizations of this process. For the sake of simplicity however, we will prove all convergence theorems in the case $N = 1$: extending those results to the general case simply amounts to replacing T with NT in the resulting error bounds.

6.3.2 Sparse estimator for the transition matrix

The transition estimator presented here is a straightforward generalization of the one used by Rao et al. (2017a). The lag-h covariance matrix of the VAR process X_t is given by the Yule-Walker recursion (see Lemma 6.A.1):

$$\Gamma_h(\theta) = \operatorname{Cov}_\theta[X_{t+h}, X_t] = \theta \Gamma_{h-1}(\theta) = \theta^h \Gamma_0(\theta) \tag{6.6}$$

We can use it to define a simple two-step procedure:

1. For a given h_0, build estimators $\widehat{\Gamma}_{h_0}$ and $\widehat{\Gamma}_{h_0+1}$ of the covariances Γ_{h_0} and Γ_{h_0+1}.
2. Use them to approximate the transition matrix by inverting Equation (6.6).

A simple inversion technique uses the Moore-Penrose pseudoinverse (just in case $\widehat{\Gamma}_{h_0}$ is singular):

$$\widehat{\theta}^{\text{dense}} = \widehat{\Gamma}_{h_0+1} \widehat{\Gamma}^\dagger_{h_0}. \tag{6.7}$$

The problem with this procedure is that is does not guarantee sparsity of $\widehat{\theta}$. To obtain a sparse result, we follow Han, Lu, and Liu (2015) and cast Equation (6.6) as a soft constraint enforcing

proximity between $\widehat{\Gamma}_{h_0+1}$ and $\widehat{\theta}\widehat{\Gamma}_{h_0}$. This amounts to solving the following constrained optimization problem:

$$\widehat{\theta} \in \operatorname*{argmin}_{\theta \in \mathbb{R}^{D\times D}} \|\mathrm{vec}(\theta)\|_1 \quad \text{subject to} \quad \|\theta\widehat{\Gamma}_{h_0} - \widehat{\Gamma}_{h_0+1}\|_{\max} \leq \lambda. \tag{6.8}$$

Here $\|\mathrm{vec}(\cdot)\|_1$ denotes the sum of the absolute values of all the coefficients of a matrix, while $\|\cdot\|_{\max}$ is the maximum of these absolute values. Given that both of these norms are piecewise linear, the problem of Equation (6.8) can be reformulated as an LP. It can even be decomposed along each dimension, which allows for an efficient and parallel solution procedure. The only thing left to do is decide how to estimate the covariance matrices Γ_h.

The covariance estimator we use is a variant of the empirical covariance. Since $Y_t = \Pi_t X_t + \eta_t$ where η_t is zero-mean, a natural proxy for X_t is obtained by inverting the sampling operator: $\widehat{X}_t = \Pi_t^\dagger Y_t$. It would therefore seem logical to build an estimator of Γ_h by plugging this proxy into the empirical covariance between X_{t+h} and X_t. However, in order for this idea to work, we must make two small adjustments.

To account for the random sampling, the plug-in empirical covariance must be scaled elementwise by a matrix $S(h) = \mathbb{E}[\pi_{t+h}\pi_t^\top]$. Intuitively, since $\widehat{X}_{t+h}\widehat{X}_t^\top$ has a fraction p^2 of nonzero coefficients, we need to divide it by something close to p^2 to get an unbiased covariance estimator. Furthermore, to account for the observation noise, we must incorporate an additive correction $-\omega^2 I$. This correction becomes unnecessary for $h \geq 1$ since the observation noise η_t is independent* across time.

In conclusion, we obtain the following covariance estimator:

$$\widehat{\Gamma}_h = \frac{1}{S(h)} \odot \frac{1}{T-h} \sum_{t=1}^{T-h} \left(\Pi_{t+h}^\dagger Y_{t+h}\right)\left(\Pi_t^\dagger Y_t\right)^\top - \mathbb{1}_{\{h=0\}}\omega^2 I. \tag{6.9}$$

The coefficients of the scaling matrix $S(h)$ are computed in Lemma 6.A.4.

6.4 Lower and upper bound on the estimation error

We now have the necessary background to formulate our theoretical results. In all the following statements (and their proofs), the letter c denotes a universal positive constant, which may change from one line to the next but never depends on any varying problem parameters. More specifically, statements involving it should always be understood as "there exists $c > 0$ such that"...

6.4.1 Main theorems

We start by bounding the non-asymptotic error of the estimator we just introduced.

Theorem 6.4.1 (Error upper bound)**.** *Consider the partially-observed VAR model defined in Section 6.3.1. We use the estimator $\widehat{\theta}$ of Section 6.3.2 with $h_0 = 0$, and we suppose that T is "large enough", as specified by Equations (6.20) and (6.23). Let us define*

$$\gamma_u(\theta) = \frac{\|\theta\|_\infty + 1}{(1 - \|\theta\|_2)^2} \frac{\sigma_{\max}^2 + \omega^2}{\|\Gamma_0(\theta)^{-1}\|_1^{-1}} \qquad \text{and} \qquad q_u = \min\{p, 1-b\} \leq p. \tag{6.10}$$

Then there is a value of λ such that the following upper bound holds with probability at least $1-\delta$:

$$\|\widehat{\theta} - \theta\|_\infty \leq c\frac{\gamma_u(\theta)s}{\sqrt{Tpq_u}}\sqrt{\log(D/\delta)}. \tag{6.11}$$

Proof. The argument combines discrete and continuous concentration inequalities, to account for both the Bernoulli sampling and the Gaussian noise. More precisely, we exploit a recent Chernoff bound that applies to non-reversible Markov chains, and we plug it into a conditional version of the Hanson-Wright inequality that we derived specifically for our purposes. See Appendix 6.A for more details. □

We now move on to a minimax lower bound which is estimator-independent, and quantifies the intrinsic difficulty of our statistical problem. The term minimax means that we study the probability of making an error of magnitude ζ, when we pick the best possible estimator $\widehat{\theta}$ and nature replies by choosing the worst possible parameter θ:

$$\mathfrak{P}(\zeta) = \inf_{\widehat{\theta}} \sup_{\theta \in \Theta_s} \mathbb{P}_\theta \left[\|\widehat{\theta} - \theta\|_\infty \geq \zeta \right]. \tag{6.12}$$

More precisely, we want to find a threshold ζ such that the probability of exceeding it is non-negligible, for instance $\mathfrak{P}(\zeta) \geq \frac{1}{2}$. The evolution of this threshold will tell us how the error behaves with respect to the various problem parameters.

Theorem 6.4.2 (Error lower bound)**.** *Consider the partially-observed VAR model defined in Section 6.3.1. We suppose that T is "large enough", as specified by Equations (6.25) and (6.27). Let us define*

$$\gamma_\ell = (1-\vartheta)^{3/2} \frac{\sigma_{\min}^2 + \omega^2}{\sigma_{\max}^2} \qquad \text{and} \qquad q_\ell = \max\{1-b, 2p-(1-b)\} \geq p. \tag{6.13}$$

Then the following minimax lower bound holds:

$$\inf_{\widehat{\theta}} \sup_{\theta \in \Theta_s} \mathbb{P}_\theta \left[\|\widehat{\theta} - \theta\|_\infty \geq c \frac{\gamma_\ell s}{\sqrt{T p q_\ell}} \right] \geq \frac{1}{2}. \tag{6.14}$$

Proof. The argument is based on an information-theoretical result known as Fano's inequality. To apply it, we need to upper-bound the Kullback-Leibler (KL) divergence between the distributions $\mathbb{P}_{\theta_0}(\Pi, Y)$ and $\mathbb{P}_{\theta_1}(\Pi, Y)$, where θ_0 and θ_1 are sufficiently far apart. See Appendix 6.B for more details. □

6.4.2 Influence of the problem parameters

Let us now compare the error bounds of Theorems 6.4.1 and 6.4.2. Our first remark is that the sparsity s and time T play exactly the same roles in both bounds, which shows that the dependency of the error in $s/\sqrt{T}$ is optimal. The lower bound does not involve the state dimension D, but this quantity shows up as $\sqrt{\log D}$ in the upper bound, so its exact influence remains unproven.

The sampling parameters appear as $1/\sqrt{pq_u}$ in the upper bound, whereas the lower bound scales as $1/\sqrt{pq_\ell}$ instead. This means that we have not proven the optimality of either bound with respect to p or b. However, it is reassuring to note that there is no conflict between them since $q_\ell \geq p \geq q_u$. Furthermore, when $a = 1 - b = p$ (that is, when Markov sampling boils down to independent sampling), both bounds simplify into the $1/p$ dependency we would expect (since $q_u = q_\ell = p$). So in the case of independent sampling, $1/p$ is indeed the optimal rate.

The ℓ_2 norm of the transition matrix plays opposite roles on each side. In the lower bound, $1-\vartheta = 1 - \max_{\theta \in \Theta_s} \|\theta\|_2$ appears in the numerator, whereas in the upper bound, $1 - \|\theta\|_2$ appears in the denominator. It is likely that these dependencies are suboptimal, but at least they are compatible

with one another: as $\|\theta\|_2 \to 1$, that is, as the VAR process becomes unstable, the lower bound tends to 0 and the upper bound to $+\infty$. This is a reflection of the fact that our proofs make heavy use of the distance between θ and the unit sphere, which means they become meaningless when θ gets too large. Note that many previous works on VAR estimation make use of the spectral radius or spectral norm of θ. While Simchowitz et al. (2018) is among the few exceptions, their assumptions are much stronger than ours (no transition sparsity, full observations). They also use a different estimator, and their bound contains some terms that ours does not need.

The variances Σ and ω^2 are involved in γ_ℓ for the lower bound, and in $\gamma_u(\theta)$ for the upper bound. In both cases, the ratio γ tells us whether the underlying process is large enough to be detected among the noise. Roughly speaking, the magnitude of X_t is related to the spectrum of Σ, while the magnitude of Y_t is related to the spectrum of $\Sigma + \omega^2 I$. If the latter is significantly larger than the former, recovering X_t (and thus θ) is a hopeless endeavor.

To simplify the comparison, let us assume in this discussion that $\Sigma = \sigma^2 I$, and that θ commutes with its transpose. Then we have $\|\Gamma_0^{-1}(\theta)\|_1^{-1} = \left\|\left(\sigma^2(I - \theta\theta^\top)^{-1}\right)^{-1}\right\|_1^{-1} = \sigma^2\|I - \theta\theta^\top\|_1^{-1}$, and we can give a simpler expression of γ_ℓ and $\gamma_u(\theta)$:

$$\gamma_u(\theta) = \frac{(\|\theta^\top\|_1 + 1)\|I - \theta\theta^\top\|_1}{(1-\vartheta)^2}\frac{\sigma^2 + \omega^2}{\sigma^2} \qquad \gamma_\ell = (1-\vartheta)^{3/2}\frac{\sigma^2 + \omega^2}{\sigma^2}.$$

We recognize the same dependency in both bounds, namely $\gamma \propto 1 + \frac{\sigma^2}{\omega^2}$. Lemma 6.4.3 gives a heuristic argument linking this functional form to the asymptotic behavior of the MLE.

Lemma 6.4.3 (Heuristic optimality of the signal-to-noise ratio)**.** *In the one-dimensional setting with full observations, the dependency of the error in $1 + \frac{\sigma^2}{\omega^2}$ is "coherent" with the asymptotic behavior of the MLE.*

Proof. Let us consider the case where $D = 1$ and $p = 1$, since we are mainly interested in the role of the parameters σ^2 and ω^2. In this case, Theorem 6.4.2 argues that the error of any estimator should grow at least like $\gamma_\ell = 1 + \frac{\omega^2}{\sigma^2}$. We also note that in this simple scenario, Theorem 6.4.1 states that $\gamma_u \propto \gamma_\ell$.

We will compare this to the asymptotic error of the Maximum Likelihood Estimator (MLE) $\widehat{\theta}$, which (for well-behaved models) is given by the inverse of the Fisher information matrix. To make this statement more precise, we will invoke Douc, Moulines, and D. Stoffer (2013, Proposition 2.14). Let us verify the conditions:

- The process is stable, i.e. $\rho(\theta) < 1$. We made sure of that by assuming $\|\theta\|_2 \leq \vartheta < 1$.
- The sampling matrix Π_t is constant across time. Although this assumption is not essential, it is true here since $p = 1$ and $D = 1$ hence $\Pi_t = I_1$.
- The model has the smallest possible dimension.
- The true parameter θ is identifiable and does not lie on the boundary of Θ_s. Identifiability is easily deduced from Lemma 6.A.1 by observing that $\theta = \Gamma_1(\theta)\Gamma_0(\theta)^{-1}$ can be entirely deduced from distribution moments.

Since all of these prerequisites hold here, Douc, Moulines, and D. Stoffer (2013, Proposition 2.14) gives us a Central Limit Theorem for the MLE of linear Gaussian models:

$$\sqrt{T}(\widehat{\theta} - \theta) \xrightarrow[T\to\infty]{\mathcal{L}} \mathcal{N}(0, \mathcal{I}_\infty(\theta)^{-1}) \quad \text{where} \quad \mathcal{I}_\infty(\theta) = \lim_{T\to\infty}\frac{\mathcal{I}_T(\theta)}{T}.$$

We only have to compute the Fisher information matrix $\mathcal{I}_T(\theta)$. The covariance matrix of Y is given by Lemma 6.B.1, but in our case the sampling matrix is constant, and we obtain the simpler (unconditional) result

$$\mathrm{Cov}_\theta[Y] = (\sigma^2 + \omega^2) I_T + R(\theta),$$

where the residual $R(\theta)$ is of order 1 in θ. Indeed, our simplifying assumptions imply $\Gamma_0(\theta) = \frac{\sigma^2}{1-\theta^2}$ and therefore

$$R(\theta) = \frac{\sigma^2}{1-\theta^2} \begin{pmatrix} \theta^2 & \theta^1 & \theta^2 & \cdots \\ \theta^1 & \theta^2 & \theta^1 & \\ \theta^2 & \theta^1 & \theta^2 & \\ \vdots & & & \ddots \end{pmatrix} \qquad \partial_\theta R(\theta) = \sigma^2 \begin{pmatrix} 0 & 1 & 0 & \cdots \\ 1 & 0 & 1 & \\ 0 & 1 & 0 & \\ \vdots & & & \ddots \end{pmatrix} + \mathcal{O}(\theta).$$

The Fisher information of Y with respect to θ has an explicit formula (Malagò and Pistone 2015, Section 3.5):

$$\begin{aligned} \mathcal{I}_T(\theta) &= \frac{1}{2} \mathrm{Tr}\left[\mathrm{Cov}_\theta[Y]^{-1} \partial_\theta \mathrm{Cov}_\theta[Y] \, \mathrm{Cov}_\theta[Y]^{-1} \partial_\theta \mathrm{Cov}_\theta[Y] \right] \\ &= \frac{1}{2} \mathrm{Tr}\left[\left(I + \frac{R(\theta)}{\sigma^2 + \omega^2} \right)^{-1} \frac{\partial_\theta R(\theta)}{\sigma^2 + \omega^2} \left(I + \frac{R(\theta)}{\sigma^2 + \omega^2} \right)^{-1} \frac{\partial_\theta R(\theta)}{\sigma^2 + \omega^2} \right]. \end{aligned}$$

If assume θ is small and perform a Taylor expansion, we get:

$$\mathcal{I}_T(\theta) \approx \frac{1}{2(\sigma^2 + \omega^2)^2} \mathrm{Tr}\left[(\partial_\theta R(\theta))^2 \right].$$

Incidentally, we also note that at the lowest order in θ,

$$\mathrm{Tr}[(\partial_\theta R(\theta))^2] = \|\partial_\theta R(\theta)\|_F^2 \approx 2\sigma^4 (T-1).$$

Which gives us an approximate information matrix for T steps:

$$\mathcal{I}_T(\theta) \approx \frac{\mathrm{Tr}[(\partial_\theta R(\theta))^2]}{2(\sigma^2 + \omega^2)^2} \approx \frac{T}{2} \left(\frac{\sigma^2}{\sigma^2 + \omega^2} \right)^2.$$

Taking the temporal limit yields:

$$\mathcal{I}_\infty(\theta) = \lim_{T \to \infty} \frac{\mathcal{I}_T(\theta)}{T} \approx \frac{1}{2} \left(\frac{\sigma^2}{\sigma^2 + \omega^2} \right)^2.$$

In conclusion, this informal analysis reveals an asymptotic error equivalent to

$$\frac{1}{\sqrt{T}} \sqrt{\mathcal{I}_\infty(\theta)^{-1}} \approx \frac{\sqrt{2}}{\sqrt{T}} \left(1 + \frac{\omega^2}{\sigma^2} \right),$$

which is coherent with the dependency we identified in Theorem 6.4.2.

□

6.4.3 Extension to VAR processes of higher order

Although our results only apply to state-space models based on an underlying VAR process of order 1, we could try to extend them to the more general case of VAR(K) processes. Just for this Section, suppose X_t is no longer given by Equation (6.3), but instead satisfies:

$$X_t = \theta_1 X_{t-1} + \theta_2 X_{t-2} + ... + \theta_K X_{t-K} + \varepsilon_t.$$

Then we can represent this as a VAR(1) process using augmented variables (Lütkepohl 2005). Indeed, observe that defining $\widetilde{X}_t = \begin{pmatrix} X_t & X_{t-1} & \cdots & X_{t-K+1} \end{pmatrix}^\top$ and $\widetilde{\varepsilon}_t = \begin{pmatrix} \varepsilon_t & 0 & \cdots & 0 \end{pmatrix}^\top$ yields

$$\widetilde{X}_t = \widetilde{\theta}\widetilde{X}_{t-1} + \widetilde{\varepsilon}_t \qquad \text{with} \qquad \widetilde{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_{K-1} & \theta_K \\ I_D & 0 & \cdots & 0 & 0 \\ 0 & I_D & & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I_D & 0 \end{bmatrix}.$$

Unfortunately, by this reasoning, the Markov sampling mechanism that generates Π_t gives rise to a new distribution for $\widetilde{\Pi}_t$ which is no longer part of the same family. Indeed, the augmented sampling process $\widetilde{\Pi}_t$ is still Markovian but with a memory of size K instead of 1. Therefore, the adaptation would require new arguments, and we leave it for future work.

6.5 Numerical experiments

We now illustrate the theory outlined above on simulated data. Our code was written in Julia (Bezanson et al. 2017), linear optimization problems were modeled using `JuMP.jl`[2] (Dunning, Huchette, and Lubin 2017) and solved with the `HiGHS`[3] solver (Huangfu and Hall 2018).

6.5.1 Data generation

Simulating a partially-observed VAR process with known transition matrix θ allows us to compute the estimation error $\|\widehat{\theta} - \theta\|_\infty$ and study the influence of parameters such as T, D, s, p, ω, *etc.* Real values for θ were drawn using independent standard Gaussian distributions for each coefficient, and then normalized to satisfy $\|\theta\|_2 = \vartheta = \frac{1}{2}$. To simplify comparison with the theoretical bounds, we used a diagonal innovation covariance $\Sigma = \sigma^2 I$ and set the sampling parameters to $a = 1 - b = p$, which amounts to independent sampling (except for the experiment that focuses specifically on the influence of b). When not mentioned explicitly, all other parameters are equal to their default values given below (we assume ω is known):

$$T = 10000 \qquad D = 5 \qquad \sigma = 1.0 \qquad \omega = 0.1 \qquad p = 1.0.$$

Most of the simulations are run in a dense estimation scenario. For those that require the sparse procedure, selecting a good regularization parameter λ is paramount: indeed, Theorem 6.4.1 is only valid for a specific value of λ (which is not known in practice, but we can hope to approximate this near-optimal choice).

[2] https://github.com/jump-dev/JuMP.jl
[3] https://github.com/ERGO-Code/HiGHS

A standard way to tune λ would be cross-validation. However, evaluating a choice of λ (and the resulting estimate $\widehat{\theta}$) requires inferring the hidden state sequence X_t from the observations Y_t. If the projection matrices Π_t were deterministic, the inference could be performed with Kalman filtering (Kalman 1960), but since they are stochastic, the distribution of (X, Y) is no longer jointly Gaussian and the justification behind the Kalman filter breaks down. Finding an appropriate inference method in our setting will be the topic of future studies.

In the meantime, to tune λ, we suppose that the sparsity level of the real transition matrix θ is known. We then use this target sparsity $\widehat{s}$ to guide a dichotomy search on λ, until we find a transition matrix estimate $\widehat{\theta}$ whose row sparsity level is sufficiently close to $\widehat{s}$.

6.5.2 Results

The main results are presented on Figure 6.1. All plots except Figure 6.1f have the estimation error $\|\widehat{\theta} - \theta\|_\infty$ on their y-axis, and some parameter of interest on their x-axis. The axes are displayed with logarithmic scaling, in order to highlight the exponent of the dependencies. Each point corresponds to one run of the algorithm, aimed at estimating a single random value of θ. When a straight line is added to a scatter plot, it is the result of a Theil-Sen regression (Sen 1968) applied to the points of the same color: its slope is denoted by α in the legend.

Figure 6.1a confirms that the error decreases as $1/\sqrt{T}$. This is only true because the sampling probability p remains constant. If instead we had a limited observation budget but an increasing temporal precision, we would have $p \propto 1/T$, in which case the error would increase as $\sqrt{T}$ instead of decreasing.

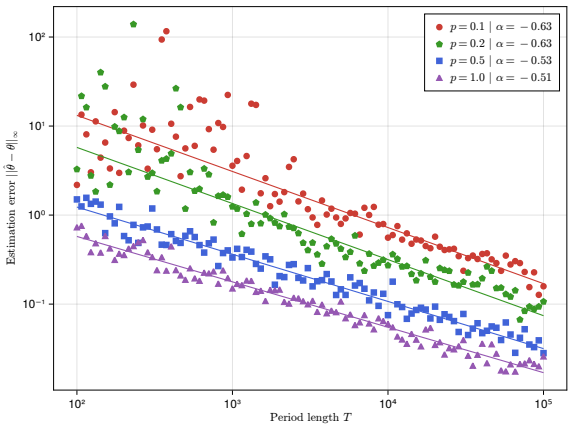
Figure 6.1b exhibits three clearly identifiable regimes with respect to the noise variance. In the first one, corresponding to $\omega/\sigma \ll 1$, the error remains small and constant. Then, the error increases when $\omega/\sigma \simeq 1$. In the third phase, corresponding to $\omega/\sigma \gg 1$, the error remains high and volatile. This is consistent with the theoretical dependency in $1 + \omega^2/\sigma^2$.

Figure 6.1c compares the respective benefits of sparse and dense estimation by increasing the ambient dimension D while keeping the true sparsity level s constant. The error for $\widehat{\theta}^{\text{dense}}$ scales linearly with D, while its sparse counterpart $\widehat{\theta}$ achieves a much slower error growth (perhaps related to the $\sqrt{\log D}$ term). As a side note, the fact that the error grows with D in the dense case is not surprising. Indeed, we measure it with the ℓ_∞ operator norm, which scales with the dimension of the matrix.

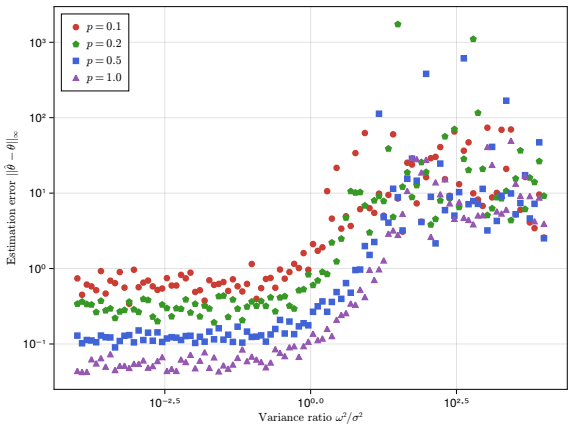
Figure 6.1d takes the opposite perspective by increasing the number of nonzero coefficients in a space of fixed dimension. In this case, the theory predicts that the error should scale linearly with s, but the slope we observe is below 1. Our interpretation is that the function $\gamma_u(\theta)$ also depends on the sparsity level in complicated ways through θ, especially since the real values are renormalized to satisfy $\|\theta\|_2 = \frac{1}{2}$.

Figure 6.1e shows that the error evolves as $1/p$, which is consistent with our upper bound. It is also informative w.r.t. the choice of h_0. Choosing $h_0 = 0$ means we need to know ω to perform estimation. If this parameter is unknown, we can choose $h_0 \geq 1$, which leads to a much higher variance of the estimator (this is not visible in our results since we wrote the proof in the case where $h_0 = 0$). An alternate solution would be to keep $h_0 = 0$ and plug in a guess such as $\omega = 0$, effectively trading lower variance for a higher bias.

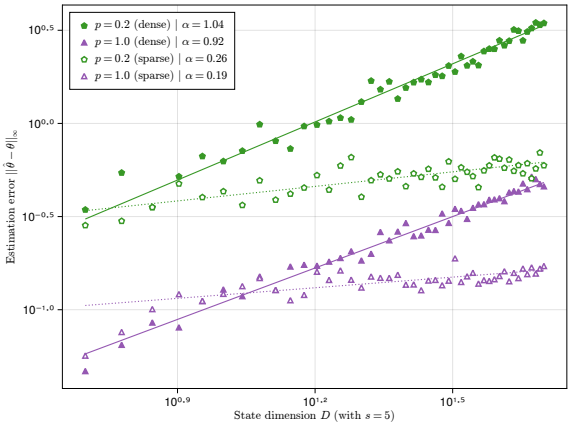
Figure 6.1f takes a closer look at the role of the Markov sampling parameter b. The white region corresponds to values of b for which there is no a such that $p = a/(a + b)$. On this logarithmic heatmap, we see regularly-spaced and nearly vertical contour lines, which is consistent with a



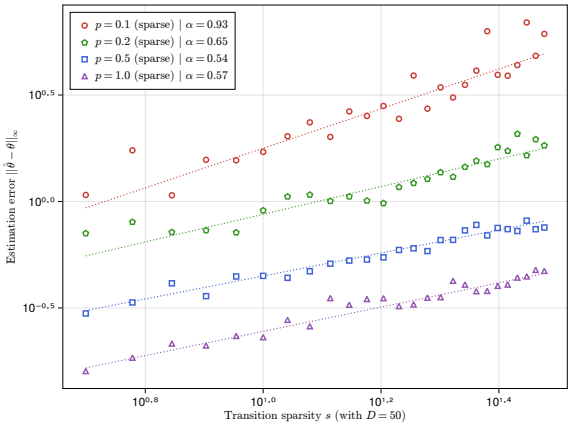
(a) Influence of T



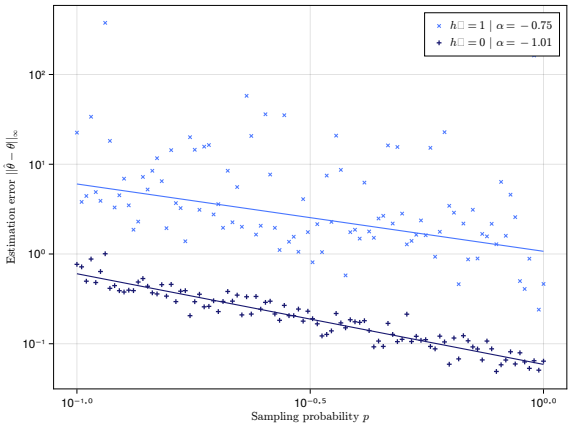
(b) Influence of ω



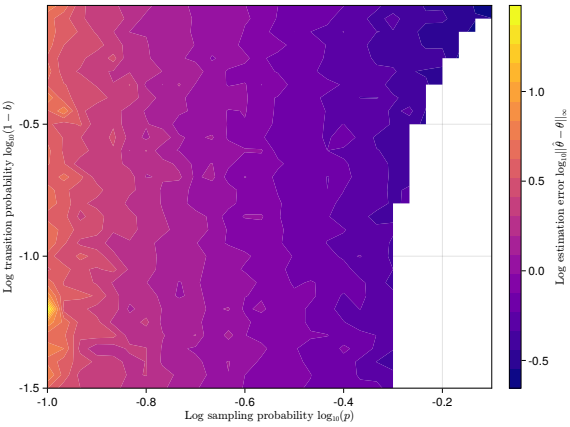
(c) Influence of D with fixed s



(d) Influence of s with fixed D



(e) Influence of p and h_0



(f) Influence of (p, b)

Figure 6.1: Impact of model parameters on the estimation error

convergence rate of $1/p$ that does not depend on b. We conjecture that $1/p$ is the true order of magnitude for the optimal error, and that the dependencies $1/\sqrt{pq_u}$ and $1/\sqrt{pq_\ell}$ from our Theorems could be refined and brought together with a more careful theoretical analysis.

6.A Proof of the estimator's convergence rate

Here we present the detailed proof of Theorem 6.4.1.

6.A.1 Overview

The main steps of the argument are the following:

1. Prove the Yule-Walker Equation (6.6) and deduce an expression for the covariance matrix of X (Lemmas 6.A.1 and 6.A.2).
2. Justify the formula of Equation (6.9) for $\widehat{\Gamma}_h$ by showing that it defines an unbiased estimator of Γ_h (Lemmas 6.A.3 and 6.A.4).
3. Fixing two indices d_1 and d_2, rewrite $(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}$ using quadratic forms $g_a^\top \Psi_a^\top L \Psi_b g_b$ of standard Gaussian vectors (Lemma 6.A.5).
4. Control the deviation of the matrix L using discrete concentration inequalities (Lemmas 6.A.7, 6.A.8, 6.A.9, 6.A.10 and 6.A.11).
5. Apply a conditional version of the Hanson-Wright inequality (Lemma 6.A.6) to the quadratic forms $g_a^\top \Psi_a^\top L \Psi_b g_b$ (Lemma 6.A.12).
6. Obtain a high-probability control on $\|\widehat{\Gamma}_h - \Gamma_h\|_{\max}$ with a union bound (Lemma 6.A.14).
7. Deduce the error of $\widehat{\theta}$ from the error of $\widehat{\Gamma}_{h_0}$ and $\widehat{\Gamma}_{h_0+1}$ by drawing inspiration from Han, Lu, and Liu (2015) (Lemmas 6.A.15 and 6.A.16).

6.A.2 Covariance matrices

The Yule-Walker equation is a direct consequence of the VAR recursion, as can be seen from this Lemma.

Lemma 6.A.1 (VAR covariance matrices)**.** *The autocovariance matrices of the stationary VAR process defined by Equation (6.3) have the following expressions:*

$$\Gamma_0(\theta) = \mathrm{Cov}_\theta[X_t] = \sum_{k=0}^{\infty} \theta^k \Sigma (\theta^k)^\top$$
$$\Gamma_h(\theta) = \mathrm{Cov}_\theta[X_{t+h}, X_t] = \theta^h \Gamma_0(\theta).$$

Proof. We start by noting that according to Equation (6.3), the stacked vector $X = (X_t)_{t\in[T]}$ follows a TD-dimensional centered multivariate Gaussian distribution. The covariance matrix of X_t can be deduced from the recursion:

$$\Gamma_0(\theta) = \mathrm{Cov}_\theta[X_t] = \mathrm{Cov}_\theta[\theta X_{t-1} + \varepsilon_t] = \theta\, \mathrm{Cov}_\theta[X_{t-1}]\theta^\top + \Sigma = \theta \Gamma_0(\theta)\theta^\top + \Sigma.$$

There is a unique stationary solution:

$$\Gamma_0(\theta) = \sum_{k=0}^{\infty} \theta^k \Sigma (\theta^k)^\top.$$

The covariance matrix between X_{t+h} and X_t is obtained similarly:

$$\begin{aligned}\Gamma_h(\theta) &= \mathrm{Cov}_\theta[X_{t+h}, X_t] = \mathbb{E}[X_{t+h} X_t^\top] = \mathbb{E}[(\theta X_{t+h-1} + \varepsilon_{t+h}) X_t^\top] \\ &= \theta \, \mathrm{Cov}_\theta[X_{t+h-1}, X_t] = \theta^h \, \mathrm{Cov}_\theta[X_t, X_t] = \theta^h \Gamma_0(\theta).\end{aligned}$$

And $\mathrm{Cov}_\theta[X_t, X_{t+h}] = \mathrm{Cov}_\theta[X_{t+h}, X_t]^\top$. In other words, we just proved that

$$\mathrm{Cov}_\theta[X] = \begin{bmatrix} \Gamma_0(\theta) & \Gamma_0(\theta)(\theta^1)^\top & \Gamma_0(\theta)(\theta^2)^\top & \cdots & \Gamma_0(\theta)(\theta^{T-1})^\top \\ \theta^1\Gamma_0(\theta) & \Gamma_0(\theta) & \Gamma_0(\theta)(\theta^1)^\top & & \\ \theta^2\Gamma_0(\theta) & \theta^1\Gamma_0(\theta) & \Gamma_0(\theta) & & \\ \vdots & & & \ddots & \\ \theta^{T-1}\Gamma_0(\theta) & & & & \Gamma_0(\theta) \end{bmatrix}$$

□

The following result will come in handy later.

Lemma 6.A.2 (Norm of $\Gamma_0(\theta)$)**.** *The covariance matrix $\Gamma_0(\theta)$ satisfies*

$$\|\Gamma_0(\theta)\|_2 \leq \frac{\sigma_{\max}^2}{1 - \vartheta^2}$$

Proof. By Lemma 6.A.1,

$$\|\Gamma_0(\theta)\|_2 \leq \sum_{k=0}^{\infty} \|\theta^k \Sigma (\theta^k)^\top\|_2 \leq \sum_{k=0}^{\infty} \|\theta\|_2^k \|\Sigma\|_2 \|\theta\|_2^k = \frac{\|\Sigma\|_2}{1 - \|\theta\|_2^2} \leq \frac{\sigma_{\max}^2}{1 - \vartheta^2}.$$

□

6.A.3 Construction of the covariance estimator

Now we justify the construction of our covariance estimator. Let $h_0 = 0$: for most of the proof, we fix a lag value $h \in \{h_0, h_0 + 1\} = \{0, 1\}$.

Lemma 6.A.3 (Bias of the covariance estimator)**.** *The estimator $\widehat{\Gamma}_h$ given by Equation (6.9) for the covariance matrix Γ_h is unbiased.*

Proof. First, let us remember that since $\Pi_t = \mathrm{diag}(\pi_t)$ is diagonal and binary, we also have $\Pi_t^\dagger = \Pi_t^\top = \Pi_t$. By Equation (6.5),

$$\begin{aligned}(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top &= \Pi_{t+h}^\dagger (\Pi_{t+h} X_{t+h} + \eta_{t+h})(X_t^\top \Pi_t^\top + \eta_t^\top)(\Pi_t^\dagger)^\top \\ &= \mathrm{diag}(\pi_{t+h}) \left(X_{t+h} X_t^\top + X_{t+h} \eta_t^\top + \eta_{t+h} X_t^\top + \eta_{t+h} \eta_t^\top \right) \mathrm{diag}(\pi_t).\end{aligned} \tag{6.15}$$

Taking the conditional expectation and removing the cross-product terms (by independence of X and Π), we get:

$$\mathbb{E}[(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top|\Pi] = \operatorname{diag}(\pi_{t+h})\left(\mathbb{E}[X_{t+h}X_t^\top] + \mathbb{E}[\eta_{t+h}\eta_t^\top]\right)\operatorname{diag}(\pi_t).$$

Since $\mathbb{E}\left[X_{t+h}X_t^\top\right] = \Gamma_h$ and $\mathbb{E}[\eta_{t+h}\eta_t] = \mathbb{1}_{\{h=0\}}\omega^2 I$, we are left with:

$$\mathbb{E}[(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top|\Pi] = (\pi_{t+h}\pi_t^\top) \odot \Gamma_h + \mathbb{1}_{\{h=0\}}\omega^2 \operatorname{diag}(\pi_t)$$

where $\odot$ is the elementwise Hadamard product. We now take the expectation w.r.t. Π:

$$\mathbb{E}[(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top] = \mathbb{E}[\pi_{t+h}\pi_t^\top] \odot \Gamma_h + \mathbb{1}_{\{h=0\}}\omega^2\mathbb{E}[\operatorname{diag}(\pi_t)].$$

Dividing elementwise by the scaling matrix $S(h) = \mathbb{E}[\pi_{t+h}\pi_t^\top]$, we get

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{\mathbb{E}[\pi_{t+h}\pi_t^\top]} \odot (\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top\right] &= \Gamma_h + \mathbb{1}_{\{h=0\}}\omega^2\mathbb{E}[\operatorname{diag}(\pi_t)] \odot \frac{1}{\mathbb{E}[\pi_t\pi_t^\top]} \\
&= \Gamma_h + \mathbb{1}_{\{h=0\}}\omega^2 \operatorname{diag}\left(\frac{\mathbb{E}[\pi_t]}{\mathbb{E}[\pi_t^2]}\right) \\
&= \Gamma_h + \mathbb{1}_{\{h=0\}}\omega^2 I
\end{aligned}$$

which shows that our estimator

$$\widehat{\Gamma}_h = \frac{1}{T-h}\sum_{t=1}^{T-h} \frac{1}{\mathbb{E}[\pi_{t+h}\pi_t^\top]} \odot (\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top - \mathbb{1}_{\{h=0\}}\omega^2 I$$

is unbiased. □

Note that since the process (Π_t) is stationary, the coefficients of $S(h)$ do not depend on t. They are computed in the next Lemma.

Lemma 6.A.4. *The second-order moments of π are given by*

$$S(h)_{d_1,d_2} = \mathbb{E}\left[\pi_{t+h,d_1}\pi_{t,d_2}\right] = \begin{cases} p^2 & \text{if } d_1 \neq d_2 \\ p & \text{if } d_1 = d_2 \text{ and } h = 0 \\ p^2 + p(1-p)(1-a-b)^h & \text{if } d_1 = d_2 \text{ and } h \geq 1 \end{cases}$$

In particular, every coefficient of the scaling matrix $S(h)$ is lower-bounded by

$$\min_{d_1,d_2,h} S(h)_{d_1,d_2} = \min\{p^2, p(1-b)\} = pq_u \quad \text{where} \quad q_u = \min\{p, 1-b\}.$$

Proof. Let $i = (t+h, d_1)$ and $j = (t, d_2)$ be two indices in $[T] \times [D]$. We have $\mathbb{E}[\pi_i] = \mathbb{E}[\pi_i^2] = p$. If $d_1 \neq d_2$, then the variables π_i and π_j belong to independent Markov chains, and thus $\mathbb{E}[\pi_i\pi_j] = p^2$. Otherwise, we have $i = (t+h, d)$ and $j = (t, d)$, which means these two variables are part of the same Markov chain. Stationarity yields

$$\mathbb{E}[\pi_i\pi_j] = \mathbb{P}(\pi_{t,d} = 1) \times \mathbb{P}(\pi_{t+h,d} = 1|\pi_{t,d} = 1) = p(P^h)_{11}.$$

When diagonalizing the transition matrix P, we see that the bottom-right coefficient of P^h is

$$(P^h)_{11} = \frac{a + b(1-a-b)^h}{a+b} = p + (1-p)(1-a-b)^h.$$

Plugging this in, we get

$$\mathbb{E}[\pi_i \pi_j] = p^2 + p(1-p)(1-a-b)^h.$$

Among all the possible values of $S(h)_{d_1,d_2}$, the smallest one is p^2 if $1-a-b \geq 0$, and $p^2 + p(1-p)(1-a-b)$ otherwise. But since

$$\begin{aligned} p + (1-p)(1-a-b) &= \frac{a}{a+b} + \frac{b}{a+b}(1-a-b) \\ &= \frac{a+b-ab-b^2}{a+b} = \frac{a(1-b)+b(1-b)}{a+b} \\ &= 1-b, \end{aligned}$$

we conclude

$$\min_{d_1,d_2,h} S(h)_{d_1,d_2} = \min\{p^2, p^2 + p(1-p)(1-a-b)\} = \min\{p^2, p(1-b)\}.$$

□

6.A.4 Gaussian concentration, episode 1

From now on, we will study the concentration of $\widehat{\Gamma}_h$, coefficient by coefficient. Let us fix two indices d_1 and d_2: our goal is to control the deviation of $(\widehat{\Gamma}_h)_{d_1,d_2}$ around its mean.

Lemma 6.A.5 (Deviation of $(\widehat{\Gamma}_h)_{d_1,d_2}$)**.** *The deviation probability for $(\widehat{\Gamma}_h)_{d_1,d_2}$ can be decomposed as follows:*

$$\begin{aligned} \mathbb{P}(|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \geq u) \leq \quad & \mathbb{P}\left(|g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\varepsilon g_\varepsilon - \mathbb{E}\left[g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\varepsilon g_\varepsilon\right]| \geq u/4\right) \\ & + \mathbb{P}\left(|g_\eta^\top \Psi_\eta^\top L \Psi_\varepsilon g_\varepsilon - \mathbb{E}\left[g_\eta^\top \Psi_\eta^\top L \Psi_\varepsilon g_\varepsilon\right]| \geq u/4\right) \\ & + \mathbb{P}\left(|g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\eta g_\eta - \mathbb{E}\left[g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\eta g_\eta\right]| \geq u/4\right) \\ & + \mathbb{P}\left(|g_\eta^\top \Psi_\eta^\top L \Psi_\eta g_\eta - \mathbb{E}\left[g_\eta^\top \Psi_\eta^\top L \Psi_\eta g_\eta\right]| \geq u/4\right) \end{aligned}$$

where the random matrix L is defined in Equation (6.16), Ψ_ε and Ψ_η are defined in Equation (6.18), and g_ε and g_η are standard Gaussian vectors.

Proof. We denote by e_d the basis vector filled with zeros except for a 1 in position d. By Equation (6.9),

$$\begin{aligned} (\widehat{\Gamma}_h + \mathbb{1}_{\{h=0\}}\omega^2 I)_{d_1,d_2} &= \frac{1}{T-h}\sum_{t=1}^{T-h}\left(\frac{1}{S(h)} \odot (\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top\right)_{d_1,d_2} \\ &= \frac{1}{T-h}\sum_{t=1}^{T-h}\frac{1}{S(h)_{d_1,d_2}}\mathrm{e}_{d_1}^\top(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top \mathrm{e}_{d_2} \\ &= \frac{1}{T-h}\sum_{t=1}^{T-h}\mathrm{Tr}\left[\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}(\Pi_{t+h}^\dagger Y_{t+h})(\Pi_t^\dagger Y_t)^\top\right] \end{aligned}$$

Equation (6.15) allows us to rewrite $(\Pi^\dagger_{t+h} Y_{t+h})(\Pi^\dagger_t Y_t)^\top$:

$$\begin{aligned}(\widehat{\Gamma}_h + \mathbb{1}_{\{h=0\}}\omega^2 I)_{d_1,d_2} = &\frac{1}{T-h}\sum_{t=1}^{T-h} X_t^\top \operatorname{diag}(\pi_t)\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}\operatorname{diag}(\pi_{t+h})X_{t+h}\\ &+\frac{1}{T-h}\sum_{t=1}^{T-h} \eta_t^\top \operatorname{diag}(\pi_t)\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}\operatorname{diag}(\pi_{t+h})X_{t+h}\\ &+\frac{1}{T-h}\sum_{t=1}^{T-h} X_t^\top \operatorname{diag}(\pi_t)\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}\operatorname{diag}(\pi_{t+h})\eta_{t+h}\\ &+\frac{1}{T-h}\sum_{t=1}^{T-h} \eta_t^\top \operatorname{diag}(\pi_t)\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}\operatorname{diag}(\pi_{t+h})\eta_{t+h}\end{aligned}$$

Let us denote by P_t the projection of $\mathbb{R}^{TD}$ keeping only the components associated with time t, i.e. such that $X_t = P_t X$ and $\eta_t = P_t \eta$. We recognize the following matrix L in all four lines of the expression above:

$$\begin{aligned}L &= \frac{1}{T-h}\sum_{t=1}^{T-h} P_t^\top \operatorname{diag}(\pi_t)\frac{\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}\operatorname{diag}(\pi_{t+h})P_{t+h}\\ &= \frac{1}{T-h}\sum_{t=1}^{T-h} P_t^\top \frac{\pi_{t+h,d_1}\pi_{t,d_2}\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top}{S(h)_{d_1,d_2}}P_{t+h}\end{aligned} \tag{6.16}$$

This leads to:

$$(\widehat{\Gamma}_h + \mathbb{1}_{\{h=0\}}\omega^2 I)_{d_1,d_2} = X^\top L X + \eta^\top L X + X^\top L \eta + \eta^\top L \eta$$

Since X and η both follow centered multivariate Gaussian distributions, we can express them as linear combinations of standard Gaussian vectors g_ε and g_η of dimension TD (indexed by the source of randomness):

$$X = \Psi_\varepsilon g_\varepsilon \qquad \text{and} \qquad \eta = \Psi_\eta g_\eta \tag{6.17}$$

where Ψ_ε and Ψ_η are the square roots of the respective covariance matrices

$$\Psi_\varepsilon = \operatorname{Cov}[X]^{1/2} \qquad \text{and} \qquad \Psi_\eta = \operatorname{Cov}[\eta]^{1/2} = \omega I. \tag{6.18}$$

We substitute X and η to get:

$$(\widehat{\Gamma}_h + \mathbb{1}_{\{h=0\}}\omega^2 I)_{d_2,d_1} = g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\varepsilon g_\varepsilon + g_\eta^\top \Psi_\eta^\top L \Psi_\varepsilon g_\varepsilon + g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\eta g_\eta + g_\eta^\top \Psi_\eta^\top L \Psi_\eta g_\eta,$$

which implies

$$\begin{aligned}(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2} = &g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\varepsilon g_\varepsilon - \mathbb{E}[g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\varepsilon g_\varepsilon]\\ &+ g_\eta^\top \Psi_\eta^\top L \Psi_\varepsilon g_\varepsilon - \mathbb{E}[g_\eta^\top \Psi_\eta^\top L \Psi_\varepsilon g_\varepsilon]\\ &+ g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\eta g_\eta - \mathbb{E}[g_\varepsilon^\top \Psi_\varepsilon^\top L \Psi_\eta g_\eta]\\ &+ g_\eta^\top \Psi_\eta^\top L \Psi_\eta g_\eta - \mathbb{E}[g_\eta^\top \Psi_\eta^\top L \Psi_\eta g_\eta].\end{aligned}$$

The union bound gives us the expected result. □

Now, our goal is to apply a Gaussian concentration inequality to these deviation probabilities. However, since L is generated by the discrete sampling process π, it is random, and so are the products $\Psi_a^\top L \Psi_b$ (where $a, b \in \{\varepsilon, \eta\}$). We thus need a conditional version of the Hanson-Wright inequality (Lemma 6.A.6), in which the following random variables will come into play:

- The spectral norm $\|\Psi_a^\top L \Psi_b\|_2$
- The Frobenius norm $\|\Psi_a^\top L \Psi_b\|_F^2$
- The shifted trace $\operatorname{Tr}(\Psi_a^\top L \Psi_b - \mathbb{E}[\Psi_a^\top L \Psi_b])$

Lemma 6.A.6 (Conditional Hanson-Wright inequality)**.** *Let A be a random square matrix such that with probability $1-\delta$,*

$$\|A\|_2 \leq M_2 \qquad \text{and} \qquad \|A\|_F^2 \leq M_F^2.$$

If X and Y are two independent standard Gaussian vectors independent of A, we have:

$$\mathbb{P}\left(|X^\top A X - \mathbb{E}[X^\top A X]| \geq u\right) \leq \delta + 2\exp\left(-c\min\left\{\frac{u^2}{M_F^2}, \frac{u}{M_2}\right\}\right) + \mathbb{P}\left(|\operatorname{Tr}(A - \mathbb{E}[A])| \geq u/2\right)$$
$$\mathbb{P}\left(|X^\top A Y - \mathbb{E}[X^\top A Y]| \geq u\right) \leq \delta + 2\exp\left(-c\min\left\{\frac{u^2}{M_F^2}, \frac{u}{M_2}\right\}\right).$$

Proof. We start with the first case. Since A is a discrete random matrix with a finite set $\mathcal{A}$ of possible values,

$$\begin{aligned}\mathbb{P}(|X^\top A X - \mathbb{E}[X^\top A X]| \geq u) &= \sum_{a \in \mathcal{A}} \mathbb{P}(|X^\top A X - \mathbb{E}[X^\top A X]| \geq u \cap A = a) \\ &= \sum_{a \in \mathcal{A}} \mathbb{P}(|X^\top a X - \mathbb{E}[X^\top A X]| \geq u \cap A = a).\end{aligned}$$

Using independence between X and A gives us

$$\mathbb{P}(|X^\top A X - \mathbb{E}[X^\top A X]| \geq u) = \sum_{a \in \mathcal{A}} \mathbb{P}(|X^\top a X - \mathbb{E}[X^\top A X]| \geq u)\mathbb{P}(A = a).$$

We now split the set of feasible values $\mathcal{A}$ into

$$\mathcal{A}_\leq = \{a \in \mathcal{A} : \|a\|_F^2 \leq M_F^2\} \quad \text{and} \quad \mathcal{A}_> = \{a \in \mathcal{A} : \|a\|_F^2 > M_F^2\}.$$

Since we assumed $\mathbb{P}(A \in \mathcal{A}_>) = \sum_{a \in A_>} \mathbb{P}(A = a) \leq \delta$, we get:

$$\mathbb{P}(|X^\top A X - \mathbb{E}[X^\top A X]| \geq u) \leq \delta + \sum_{a \in \mathcal{A}_\leq} \mathbb{P}(|X^\top a X - \mathbb{E}[X^\top A X]| \geq u)\mathbb{P}(A = a).$$

Unfortunately, Lemma A.2.10 only lets us bound

$$\mathbb{P}(|X^\top a X - \mathbb{E}[X^\top a X]| \geq u) \quad \text{and not} \quad \mathbb{P}(|X^\top a X - \mathbb{E}[X^\top A X]| \geq u)$$

(notice the change inside the expectation), which means we need an additional step. For a fixed $a \in \mathcal{A}_{\leq}$, we use independence and normality to obtain

$$\begin{aligned}\mathbb{E}[X^\top aX] - \mathbb{E}[X^\top AX] &= \mathbb{E}[\mathrm{Tr}(X^\top(a-A)X)] = \mathrm{Tr}(\mathbb{E}[XX^\top(a-A)]) \\ &= \mathrm{Tr}(\mathbb{E}[XX^\top]\mathbb{E}[a-A]) = \mathrm{Tr}(a - \mathbb{E}[A]).\end{aligned}$$

We are now ready to decompose, with the help of the union bound:

$$\begin{aligned}\mathbb{P}(|X^\top aX - \mathbb{E}[X^\top AX]| \geq u) &= \mathbb{P}\left(|X^\top aX - \mathbb{E}[X^\top aX] + \mathbb{E}[X^\top aX] - \mathbb{E}[X^\top AX]| \geq u\right) \\ &\leq \mathbb{P}\left(|X^\top aX - \mathbb{E}[X^\top aX]| \geq u/2\right) + \mathbb{P}\left(|\mathbb{E}[X^\top aX] - \mathbb{E}[X^\top AX]| \geq u/2\right) \\ &\leq 2\exp\left(-c\min\left\{\frac{u^2}{\|a\|_F^2}, \frac{u}{\|a\|_2}\right\}\right) + \mathbf{1}\left\{|\,\mathrm{Tr}(a - \mathbb{E}[A])| \geq u/2\right\}.\end{aligned}$$

This implies:

$$\begin{aligned}\mathbb{P}(|X^\top AX - \mathbb{E}[X^\top AX]| \geq u) &\leq \delta + \sum_{a \in \mathcal{A}_{\leq}} \mathbb{P}(A = a)\mathbb{P}(|X^\top aX - \mathbb{E}[X^\top AX]| \geq u) \\ &\leq \delta + \sum_{a \in \mathcal{A}_{\leq}} \mathbb{P}(A = a) \times 2\exp\left[-c\min\left\{\frac{u^2}{\|a\|_F^2}, \frac{u}{\|a\|_2}\right\}\right] \\ &\quad + \sum_{a \in \mathcal{A}_{\leq}} \mathbb{P}(A = a) \times \mathbf{1}\left\{|\,\mathrm{Tr}(a - \mathbb{E}[A])| \geq u/2\right\}.\end{aligned}$$

By definition of $\mathcal{A}_{\leq}$,

$$\begin{aligned}\mathbb{P}(|X^\top AX - \mathbb{E}[X^\top AX]| \geq u) &\leq \delta + \sum_{a \in \mathcal{A}_{\leq}} \mathbb{P}(A = a) \times 2\exp\left(-c\min\left\{\frac{u^2}{M_F^2}, \frac{u}{M_2}\right\}\right) \\ &\quad + \mathbb{P}\left(|\,\mathrm{Tr}(A - \mathbb{E}[A])| \geq u/2\right) \\ &\leq \delta + 2\exp\left(-c\min\left\{\frac{u^2}{M_F^2}, \frac{u}{M_2}\right\}\right) + \mathbb{P}\left(|\,\mathrm{Tr}(A - \mathbb{E}[A])| \geq u/2\right).\end{aligned}$$

The proof for $X^\top AY$ follows the same lines, except that we replace $\mathbb{E}[XX^\top] = I$ by $\mathbb{E}[XY^\top] = 0$, which removes the trace term in the final expression. □

6.A.5 Interlude: discrete concentration

We exploit discrete concentration results to bound the deviations of the three quantities we just mentioned, starting with the norms.

Lemma 6.A.7 (Norm reformulation for L)**.** *The spectral and Frobenius norms of L are given by*

$$\|L\|_2 = \frac{\max_{t \in [T-h]} \pi_{t+h,d_1}\pi_{t,d_2}}{(T-h)S(h)_{d_1,d_2}} \quad \textit{and} \quad \|L\|_F^2 = \frac{1}{(T-h)^2 S(h)_{d_1,d_2}} \sum_{t=1}^{T-h} \pi_{t+h,d_1}\pi_{t,d_2}.$$

Proof. We first notice that L has a block-superdiagonal structure of rank h:

$$L = \frac{1}{T-h}\sum_{t=1}^{T-h} P_t^\top L_{[t,t+h]} P_{t+h} \qquad \text{with} \qquad L_{[t,t+h]} = \frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} \mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top. \tag{6.19}$$

The spectral and Frobenius norms of such a matrix can easily be deduced from those of its blocks. Since $\|\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top\|_2 = \|\mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top\|_F = 1$ and the π_t are binary-valued, this leads to the following formulas:

$$\|L\|_2 = \frac{1}{T-h}\max_{t\in[T-h]}\|L_{[t,t+h]}\|_2 = \frac{1}{(T-h)S(h)_{d_1,d_2}}\max_{t\in[T-h]}\pi_{t+h,d_1}\pi_{t,d_2}$$
$$\|L\|_F^2 = \frac{1}{(T-h)^2}\sum_{t=1}^{T-h}\|L_{[t,t+h]}\|_F^2 = \frac{1}{(T-h)^2 S(h)_{d_1,d_2}^2}\sum_{t=1}^{T-h}\pi_{t+h,d_1}\pi_{t,d_2}.$$

□

We can bound the spectral norm for free.

Lemma 6.A.8 (Spectral norm bound for L)**.** *With probability* 1*, the spectral norm* $\|L\|_2$ *satisfies*

$$\|L\|_2 \le \frac{c}{Tpq_u}$$

Proof. Note that $S(h)_{d_1,d_2} \ge pq_u$, and since $h \in \{0,1\}$, we can state that $T-h \ge cT$. By Lemma 6.A.7, we deduce

$$\|L\|_2 = \frac{\max_{t\in[T-h]}\pi_{t+h,d_1}\pi_{t,d_2}}{(T-h)S(h)_{d_1,d_2}} \le \frac{1}{(T-h)S(h)_{d_1,d_2}} \le \frac{1}{(T-h)pq_u} \le \frac{1}{cTpq_u}.$$

□

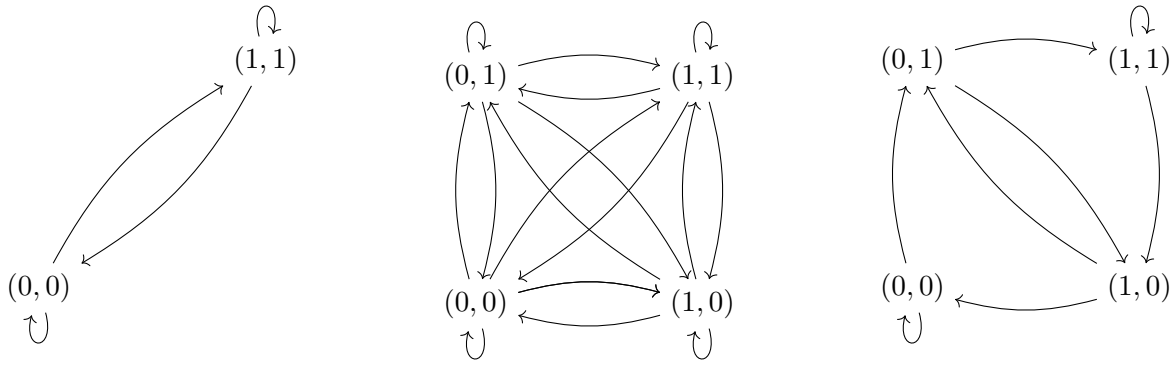
The Frobenius norm requires a little more work because of the sum it contains.

Lemma 6.A.9 (Concentration of the sampling Bernoullis)**.** *For all* $u \in [0,1]$*,*

$$\mathbb{P}\left(\left|\frac{1}{T-h}\sum_{t=1}^{T-h}\pi_{t+h,d_1}\pi_{t,d_2} - S(h)_{d_1,d_2}\right| \ge uS(h)_{d_1,d_2}\right) \le c_1\exp(-c_2u^2TS(h)_{d_1,d_2}).$$

Proof. We distinguish three cases:

- When $d_1 = d_2$ and $h = 0$, we have $\pi_{t+h,d_1} = \pi_{t,d_2}$, which is a 2-state Markov chain with transition matrix $P \otimes I$, depicted on Figure 6.2a.
- When $d_1 \neq d_2$, the couple $(\pi_{t,d_2}, \pi_{t+h,d_1})$ is a 4-state Markov chain with transition matrix $P \otimes P$ since the chains π_{t+h,d_1} and π_{t,d_2} evolve along independent dimensions. It is shown on Figure 6.2b.
- When $d_1 = d_2$ and $h \ge 1$, we must study the $(h+1)$-tuple $(\pi_{t,d_1}, \pi_{t+1,d_1}, ..., \pi_{t+h,d_1})$. It is a 2^{h+1}-state Markov chain with transition matrix $\mathcal{T}(h)$, whose non-reversible transition diagram can be seen on Figure 6.2c.



(a) $d_1 = d_2$ & $h = 0$: transition P

(b) $d_1 \neq d_2$: transition $P \otimes P$

(c) $d_1 = d_2$ & $h = 1$: transition $\mathcal{T}(1)$.

Figure 6.2: State space and transitions for the Markov chains used in the discrete concentration result

In all of these cases, our variable of interest $\pi_{t+h,d_1}\pi_{t,d_2}$ is a function of the underlying Markov chain. The relevant functions are:

$$f_1 : x \mapsto x \qquad f_2 : (x, y) \mapsto yx \qquad f_3 : (x_0, ..., x_h) \mapsto x_h x_0.$$

We note that since $\chi \leq a, b \leq 1 - \chi$, all the coefficients of P are greater than χ. Furthermore, all the coefficients of $P \otimes P$ are greater than χ^2. Finally, all the coefficients of $\mathcal{T}(h)^{h+1}$ are greater than χ^{h+1}, because all pairs of states are connected after $h+1$ steps. Let us illustrate this with $h = 1$:

$$\mathcal{T}(1) = \begin{pmatrix} 1-a & a & 0 & 0 \\ 0 & 0 & b & 1-b \\ 1-a & a & 0 & 0 \\ 0 & 0 & b & 1-b \end{pmatrix} \qquad \mathcal{T}(1)^2 = \begin{pmatrix} (1-a)^2 & a(1-a) & ab & a(1-b) \\ (1-a)b & ab & (1-b)b & (1-b)^2 \\ (1-a)^2 & a(1-a) & ab & a(1-b) \\ (1-a)b & ab & (1-b)b & (1-b)^2 \end{pmatrix}.$$

Subsequently, all the transition matrices $\mathcal{R}$ we are interested in, namely $\mathcal{R} \in \{P, P \otimes P, \mathcal{T}(h)^{h+1}\}$, satisfy the Doeblin condition with $r = h + 1$ and $\delta = \chi^{h+1}$:

$$\mathcal{R}^{h+1} \geq \chi^{h+1} \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}.$$

Since we only consider $h \in \{0, 1\}$ and since χ is fixed for our purposes, these chains fulfill the assumptions of Lemma A.2.8. We thus conclude:

$$\mathbb{P}\left(\left| \frac{1}{T-h} \sum_{t=1}^{T-h} \pi_{t+h,d_1}\pi_{t,d_2} - S(h)_{d_1,d_2} \right| \geq u S(h)_{d_1,d_2} \right) \leq c_1 \exp\left(-c_2 u^2 (T-h) S(h)_{d_1,d_2}\right).$$

We finally replace $T - h$ with cT in the exponential, leading to the result we announced. □

Based on this concentration property, we can now bound the norms of the random matrix L with high probability.

Lemma 6.A.10 (Frobenious norm bound for L)**.** *For any δ such that Equation (6.20) holds, with probability at least $1-\delta$, the Frobenius norm $\|L\|_F^2$ satisfies*

$$\|L\|_F^2 \leq \frac{c}{Tpq_u}.$$

Proof. By Lemma 6.A.9: for all $u \in [0,1]$,

$$\mathbb{P}\left(\frac{1}{T-h}\sum_{t=1}^{T-h}\pi_{t+h,d_1}\pi_{t,d_2} \geq (1+u)S(h)_{d_1,d_2}\right) \leq c_1\exp(-c_2u^2TS(h)_{d_1,d_2}).$$

We remember the expression of Lemma 6.A.7 for $\|L\|_F^2$ and notice that:

$$\begin{aligned}
&\mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{(T-h)S(h)_{d_1,d_2}}\right) \\
&\quad= \mathbb{P}\left(\frac{1}{(T-h)S(h)_{d_1,d_2}}\left(\frac{1}{T-h}\sum_{t=1}^{T-h}\frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}}\right) \geq \frac{1}{(T-h)S(h)_{d_1,d_2}}(1+u)\right) \\
&\quad\leq c_1\exp(-c_2u^2TS(h)_{d_1,d_2})
\end{aligned}$$

We finally recall that $S(h)_{d_1,d_2} \geq pq_u$ and $T-h \geq cT$, so that

$$\begin{aligned}
\mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{cTpq_u}\right) &\leq \mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{(T-h)pq_u}\right) \\
&\leq \mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{(T-h)S(h)_{d_1,d_2}}\right) \\
&\leq c_1\exp\left(-c_2u^2TS(h)_{d_1,d_2}\right) \\
&\leq c_1\exp\left(-c_2u^2Tpq_u\right).
\end{aligned}$$

All we need to make sure that $\mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{cTpq_u}\right) \leq \delta$ is to choose u such that

$$c_1\exp\left(-c_2u^2Tpq_u\right) \leq \delta \quad\iff\quad u \geq \sqrt{\frac{\log(c_1/\delta)}{c_2Tpq_u}}$$

Note that we can replace $\log(c_1/\delta)$ by a constant times $\log(1/\delta)$ to simplify expressions: this is possible as long as δ is chosen "small enough" (i.e. smaller than some universal constant). We will assume this fairly often in the rest of the proof.

For Lemma 6.A.9 to apply, we must ensure that our choice of u is smaller than 1. With the previous discussion in mind, $u \leq 1$ is implied by

$$\sqrt{\frac{\log(1/\delta)}{Tpq_u}} \leq c. \tag{6.20}$$

If this holds, then we have

$$\mathbb{P}\left(\|L\|_F^2 \geq \frac{2}{cTpq_u}\right) \leq \mathbb{P}\left(\|L\|_F^2 \geq \frac{1+u}{cTpq_u}\right) \leq \delta.$$

This yields the result we wanted. □

We now move on to studying the shifted trace of $\Psi_a^\top L \Psi_b$, which is the last ingredient we need for our application of Lemma 6.A.6.

Lemma 6.A.11 (Trace bound for the L matrices)**.** *For all $u \in [0,1]$,*

$$\mathbb{P}(|\operatorname{Tr}(\Psi_\varepsilon^\top L \Psi_\varepsilon - \mathbb{E}[\Psi_\varepsilon^\top L \Psi_\varepsilon])| \geq u) \leq c_1 \exp\left(-\frac{c_2 u^2 T p q_u}{\|\Gamma_h\|_2^2}\right)$$
$$\mathbb{P}(|\operatorname{Tr}(\Psi_\eta^\top L \Psi_\eta - \mathbb{E}[\Psi_\eta^\top L \Psi_\eta])| \geq u) \leq c_1 \exp\left(-\frac{c_2 u^2 T p q_u}{\omega^4}\right).$$

Proof. We can compute an explicit formula thanks to Equation (6.19): if $a \in \{\varepsilon, \eta\}$ then

$$\begin{aligned}
\operatorname{Tr}(\Psi_a^\top L \Psi_a) &= \operatorname{Tr}\left(\frac{1}{T-h}\sum_{t=1}^{T-h} \Psi_a^\top P_t^\top \frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} \mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top P_{t+h}\Psi_a\right) \\
&= \frac{1}{T-h}\sum_{t=1}^{T-h} \frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} \operatorname{Tr}\left(\Psi_a^\top P_t^\top \mathrm{e}_{d_2}\mathrm{e}_{d_1}^\top P_{t+h}\Psi_a\right) \\
&= \frac{1}{T-h}\sum_{t=1}^{T-h} \frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} \left(\mathrm{e}_{d_1}^\top P_{t+h}\Psi_a\Psi_a^\top P_t^\top \mathrm{e}_{d_2}\right) \\
&= \frac{1}{T-h}\sum_{t=1}^{T-h} \frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} \left((\Psi_a\Psi_a^\top)_{[t+h,t]}\right)_{d_1,d_2}
\end{aligned}$$

where $\left((\Psi_a\Psi_a^\top)_{[t+h,t]}\right)_{d_1,d_2}$ denotes the (d_1,d_2) coefficient of the $(t+h,t)$ block of $\Psi_a\Psi_a^\top$. Now is the time to look back on Equation (6.18), which tells us that both $\Psi_\varepsilon\Psi_\varepsilon^\top$ and $\Psi_\eta\Psi_\eta^\top$ are constant along their superdiagonal of rank h. We thus find that

$$\operatorname{Tr}(\Psi_\varepsilon^\top L \Psi_\varepsilon - \mathbb{E}[\Psi_\varepsilon^\top L \Psi_\varepsilon]) = (\Gamma_h)_{d_1,d_2}\left(\frac{1}{T-h}\sum_{t=1}^{T-h}\frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} - 1\right)$$
$$\operatorname{Tr}(\Psi_\eta^\top L \Psi_\eta - \mathbb{E}[\Psi_\eta^\top L \Psi_\eta]) = (\mathbb{1}_{\{h=0\}}\omega^2 I)_{d_1,d_2}\left(\frac{1}{T-h}\sum_{t=1}^{T-h}\frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} - 1\right)$$

Like before, we can apply Lemma 6.A.9: for all $u \in [0,1]$,

$$\mathbb{P}\left(\left|\frac{1}{T-h}\sum_{t=1}^{T-h}\frac{\pi_{t+h,d_1}\pi_{t,d_2}}{S(h)_{d_1,d_2}} - 1\right| \geq u\right) \leq c_1\exp(-c_2u^2TS(h)_{d_1,d_2}) \leq c_1\exp(-c_2u^2Tpq_u).$$

Since $|(\Gamma_h)_{d_1,d_2}| \leq \|\Gamma_h\|_2$ and $(\mathbb{1}_{\{h=0\}}\omega^2 I)_{d_1,d_2} \leq \omega^2$, we can deduce

$$\mathbb{P}\left(|\operatorname{Tr}(\Psi_\varepsilon^\top L \Psi_\varepsilon - \mathbb{E}[\Psi_\varepsilon^\top L \Psi_\varepsilon])| \geq u\|\Gamma_h\|_2\right) \leq c_1\exp(-c_2u^2Tpq_u)$$
$$\mathbb{P}\left(|\operatorname{Tr}(\Psi_\eta^\top L \Psi_\eta - \mathbb{E}[\Psi_\eta^\top L \Psi_\eta])| \geq u\omega^2\right) \leq c_1\exp(-c_2u^2Tpq_u)$$

which, after rescaling, yields the result we announced. □

6.A.6 Gaussian concentration, episode 2

We are now ready to apply our conditional concentration result.

Lemma 6.A.12 (Applying Hanson-Wright)**.** *Let $\delta > 0$ and $u \in [0,1]$. Assume that Equations (6.20) and (6.21) hold. Then the deviation probability for $(\widehat{\Gamma}_h)_{d_1,d_2}$ satisfies*

$$\mathbb{P}(|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \geq u) \leq 4\delta + c_1 \exp\left(-\frac{c_2 u^2 T p q_u}{\max\left\{(\|\Psi_\varepsilon\|_2^2 + \omega^2)^2, \|\Gamma_h\|_2^2, \omega^4\right\}}\right).$$

Proof. The conclusion we had reached before our discrete interlude is given by Lemma 6.A.5, and we can rewrite it as

$$\mathbb{P}(|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \geq u) \leq p_{\varepsilon\varepsilon} + p_{\eta\varepsilon} + p_{\varepsilon\eta} + p_{\eta\eta},$$

where each p_{ab} represents a deviation probability for a specific quadratic form $g_a^\top \Psi_a^\top L \Psi_b g_b$. Let us choose δ such that Equation (6.20) holds. By Lemmas 6.A.8 and 6.A.10, with probability at least $1-\delta$, the following eight inequalities occur at the same time (we use Lemma A.1.4 to split the products):

$$\begin{aligned}
\|\Psi_\varepsilon^\top L \Psi_\varepsilon\|_F^2 &\leq \frac{c\|\Psi_\varepsilon\|_2^4}{Tpq_u} & \|\Psi_\varepsilon^\top L \Psi_\varepsilon\|_2 &\leq \frac{c\|\Psi_\varepsilon\|_2^2}{Tpq_u} \\
\|\Psi_\eta^\top L \Psi_\varepsilon\|_F^2 &\leq \frac{c\|\Psi_\eta\|_2^2\|\Psi_\varepsilon\|_2^2}{Tpq_u} & \|\Psi_\eta^\top L \Psi_\varepsilon\|_2 &\leq \frac{c\|\Psi_\eta\|_2\|\Psi_\varepsilon\|_2}{Tpq_u} \\
\|\Psi_\varepsilon^\top L \Psi_\eta\|_F^2 &\leq \frac{c\|\Psi_\varepsilon\|_2^2\|\Psi_\eta\|_2^2}{Tpq_u} & \|\Psi_\varepsilon^\top L \Psi_\eta\|_2 &\leq \frac{c\|\Psi_\varepsilon\|_2\|\Psi_\eta\|_2}{Tpq_u} \\
\|\Psi_\eta^\top L \Psi_\eta\|_F^2 &\leq \frac{c\|\Psi_\eta\|_2^4}{Tpq_u} & \|\Psi_\eta^\top L \Psi_\eta\|_2 &\leq \frac{c\|\Psi_\eta\|_2^2}{Tpq_u}.
\end{aligned}$$

The spectral norm of Ψ_η is easily seen to equal $\|\Psi_\eta\|_2 = \|\omega^2 I\|_2^{1/2} = \omega$, which allows us to lighten these expressions. From there, Lemma 6.A.6 (applied with $X = g_a$, $Y = g_b$ and $A = \Psi_a^\top L \Psi_b$) provides the concentration bounds we need[4]:

$$\begin{aligned}
p_{\varepsilon\varepsilon} &\leq \delta + 2\exp\left(-cTpq_u \min\left\{\frac{(u/4)^2}{\|\Psi_\varepsilon\|_2^4}, \frac{(u/4)}{\|\Psi_\varepsilon\|_2^2}\right\}\right) + \mathbb{P}\left(|\operatorname{Tr}(\Psi_\eta^\top L \Psi_\varepsilon) - \mathbb{E}[\Psi_\varepsilon^\top L \Psi_\varepsilon])| \geq u/8\right) \\
p_{\eta\varepsilon} &\leq \delta + 2\exp\left(-cTpq_u \min\left\{\frac{(u/4)^2}{\omega^2\|\Psi_\varepsilon\|_2^2}, \frac{(u/4)}{\omega\|\Psi_\varepsilon\|_2}\right\}\right) \\
p_{\varepsilon\eta} &\leq \delta + 2\exp\left(-cTpq_u \min\left\{\frac{(u/4)^2}{\|\Psi_\varepsilon\|_2^2\omega^2}, \frac{(u/4)}{\|\Psi_\varepsilon\|_2\omega}\right\}\right) \\
p_{\eta\eta} &\leq \delta + 2\exp\left(-cTpq_u \min\left\{\frac{(u/4)^2}{\omega^4}, \frac{(u/4)}{\omega^2}\right\}\right) + \mathbb{P}\left(|\operatorname{Tr}(\Psi_\eta^\top L \Psi_\eta) - \mathbb{E}[\Psi_\eta^\top L \Psi_\eta])| \geq u/8\right).
\end{aligned}$$

The denominators inside the minima can be unified: for the left column,

$$\max\left\{\|\Psi_\varepsilon\|_2^4, \|\Psi_\varepsilon\|_2^2\omega^2, \omega^4\right\} \leq \left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right)^2,$$

[4]The additional trace terms that appear when applying Lemma 6.A.6 (as opposed to the non-conditional version of Lemma A.2.10) are absent from the papers by Rao et al. (2017a) and Rao et al. (2017b), which is why we think their upper bound proofs are incomplete.

and for the right column,

$$\max\left\{\|\Psi_\varepsilon\|_2^2, \|\Psi_\varepsilon\|_2\omega, \omega^2\right\} \le \left(\|\Psi_\varepsilon\|_2 + \omega\right)^2 \le 2\left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right).$$

This means we can upper bound each of the four minima by

$$\min\left\{\left(\frac{u/4}{\|\Psi_\varepsilon\|_2^2 + \omega^2}\right)^2, \frac{u/8}{\|\Psi_\varepsilon\|_2^2 + \omega^2}\right\}.$$

From now on, we additionally suppose that

$$\frac{u/4}{\|\Psi_\varepsilon\|_2^2 + \omega^2} \le \frac{1}{2} \tag{6.21}$$

This enables us to get rid of these minima by reducing them to the (smaller) quadratic term on the left. We end up with

$$\begin{aligned}
p_{\varepsilon\varepsilon} &\le \delta + 2\exp\left(-\frac{cu^2Tpq_u}{\left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right)^2}\right) + \mathbb{P}\left(\left|\operatorname{Tr}(\Psi_\varepsilon^\top L\Psi_\varepsilon) - \mathbb{E}[\Psi_\varepsilon^\top L\Psi_\varepsilon]\right| \ge u/8\right)\\
p_{\eta\varepsilon} &\le \delta + 2\exp\left(-\frac{cu^2Tpq_u}{\left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right)^2}\right)\\
p_{\varepsilon\eta} &\le \delta + 2\exp\left(-\frac{cu^2Tpq_u}{\left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right)^2}\right)\\
p_{\eta\eta} &\le \delta + 2\exp\left(-\frac{cu^2Tpq_u}{\left(\|\Psi_\varepsilon\|_2^2 + \omega^2\right)^2}\right) + \mathbb{P}\left(\left|\operatorname{Tr}(\Psi_\eta^\top L\Psi_\eta) - \mathbb{E}[\Psi_\eta^\top L\Psi_\eta]\right| \ge u/8\right).
\end{aligned}$$

As for the trace terms, they are taken care of by Lemma 6.A.11:

$$\begin{aligned}
\mathbb{P}\left(\left|\operatorname{Tr}(\Psi_\eta^\top L\Psi_\varepsilon) - \mathbb{E}[\Psi_\varepsilon^\top L\Psi_\varepsilon]\right| \ge u/8\right) &\le c_3\exp\left(-c_4\frac{(u/8)^2Tpq_u}{\|\Gamma_h\|_2^2}\right)\\
\mathbb{P}\left(\left|\operatorname{Tr}(\Psi_\eta^\top L\Psi_\eta) - \mathbb{E}[\Psi_\eta^\top L\Psi_\eta]\right| \ge u/8\right) &\le c_3\exp\left(-c_4\frac{(u/8)^2Tpq_u}{\omega^4}\right)
\end{aligned}$$

We plug this in and rearrange to get:

$$p_{\varepsilon\varepsilon} + p_{\eta\varepsilon} + p_{\varepsilon\eta} + p_{\eta\eta} \le 4\delta + c_1\exp\left(-\frac{c_2u^2Tpq_u}{\max\left\{(\|\Psi_\varepsilon\|_2^2 + \omega^2)^2, \|\Gamma_h\|_2^2, \omega^4\right\}}\right).$$

□

The following result will simplify the denominator inside the exponential.

Lemma 6.A.13 (Spectral norms of Ψ_ε and Γ_h)**.** *The matrices Ψ_ε and Γ_h satisfy:*

$$\|\Psi_\varepsilon\|_2^2 \le \frac{\sigma_{\max}^2}{(1-\vartheta)^2} \qquad \text{and} \qquad \|\Gamma_h\|_2 \le \frac{\vartheta^h\sigma_{\max}^2}{1-\vartheta}.$$

As a consequence,

$$\max\left\{(\|\Psi_\varepsilon\|_2^2 + \omega^2)^2, \|\Gamma_h\|_2^2, \omega^4\right\} \le \frac{(\sigma_{\max}^2 + \omega^2)^2}{(1-\vartheta)^4}$$

Proof. By Lemma 6.A.1, we can write Ψ_ε^2 as a sum of Kronecker products (one for each block). Let J_t be a matrix full of zeros, except for the subdiagonal of rank t, which is full of ones. Then we have:

$$\Psi_\varepsilon^2 = \mathrm{Cov}[X] = I \otimes \Gamma_0(\theta) + \sum_{t=1}^{T-1} \left[J_t \otimes \theta^t \Gamma_0(\theta) + J_t^\top \otimes \Gamma_0(\theta)(\theta^t)^\top \right]$$

This gives us control over its spectral norm thanks to Lemma A.1.3:

$$\begin{aligned}\|\Psi_\varepsilon\|_2^2 = \|\Psi_\varepsilon^2\|_2 &\leq \|I\|_2 \times \|\Gamma_0(\theta)\|_2 + \sum_{t=1}^{T-1} \left[\|J_t\|_2 \times \|\theta^t \Gamma_0(\theta)\|_2 + \|J_t^\top\|_2 \times \|\Gamma_0(\theta)(\theta^t)^\top\|_2 \right] \\ &\leq \|\Gamma_0(\theta)\|_2 \left(1 + 2 \sum_{t=1}^{T-1} \|\theta\|_2^t \right)\end{aligned}$$

We now use Lemma 6.A.2:

$$\|\Psi_\varepsilon\|_2^2 \leq \frac{\sigma_{\max}^2}{1-\vartheta^2} \left(1 + 2 \frac{\vartheta}{1-\vartheta} \right) = \frac{\sigma_{\max}^2}{(1-\vartheta)^2}.$$

We now turn to Γ_h with Lemmas 6.A.1 and 6.A.2:

$$\|\Gamma_h\|_2 = \|\theta^h \Gamma_0(\theta)\|_2 \leq \frac{\vartheta^h \sigma_{\max}^2}{1-\vartheta^2}.$$

In particular, we have

$$\begin{aligned}\max\left\{ (\|\Psi_\varepsilon\|_2^2 + \omega^2)^2, \|\Gamma_h\|_2^2, \omega^4 \right\} &\leq \max\left\{ \left(\frac{\sigma_{\max}^2}{(1-\vartheta)^2} + \omega^2 \right)^2, \left(\frac{\vartheta^h \sigma_{\max}^2}{1-\vartheta^2} \right)^2, \omega^4 \right\} \\ &\leq \frac{(\sigma_{\max}^2 + \omega^2)^2}{(1-\vartheta)^4}\end{aligned}$$

□

We can now control the error of the covariance estimator:

Lemma 6.A.14 (Max norm convergence rate of the covariance estimator)**.** *Let $\delta > 0$ be small enough. Assume that Equations (6.20) and (6.23) hold. Then the covariance estimator $\widehat{\Gamma}_h$ from Equation (6.9) satisfies*

$$\|\widehat{\Gamma}_h - \Gamma_h\|_{\max} \leq c \frac{\sigma_{\max}^2 + \omega^2}{(1-\vartheta)^2} \frac{\sqrt{\log(D/\delta)}}{\sqrt{T p q_u}} = \mathrm{err}(\delta)$$

with probability greater than $1 - \delta$.

Proof. Let us plug Lemma 6.A.13 into Lemma 6.A.12

$$\mathbb{P}(|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \geq u) \leq 4\delta + c_1 \exp\left(-\frac{c_2 (1-\vartheta)^4 u^2 T p q_u}{(\sigma_{\max}^2 + \omega^2)^2} \right).$$

All that is left to do is choose u such that

$$\mathbb{P}(|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \geq u) \leq 8\delta,$$

which will be true if

$$c_1 \exp\left(-\frac{c_2(1-\vartheta)^4 T p q_u}{(\sigma_{\max}^2+\omega^2)^2}u^2\right) \leq 4\delta \quad \iff \quad u \geq \sqrt{\frac{\log(c_1/4\delta)(\sigma_{\max}^2+\omega^2)^2}{c_2(1-\vartheta)^4 T p q_u}}.$$

As long as δ is small enough, we can take

$$u = c\frac{\sqrt{\log(1/\delta)}(\sigma_{\max}^2+\omega^2)}{(1-\vartheta)^2\sqrt{Tpq_u}}. \tag{6.22}$$

For Lemma 6.A.12 to apply, we must verify that $u \in [0,1]$ and that Equation (6.21) is satisfied. In other words, we have to ensure that

$$c\frac{\sqrt{\log(1/\delta)}(\sigma_{\max}^2+\omega^2)}{(1-\vartheta)^2\sqrt{Tpq_u}} \leq \min\{1, 2(\|\Psi_\varepsilon\|_2^2+\omega^2)\}$$

Using Lemma 6.A.13, this is implied by the condition

$$\frac{\sqrt{\log(1/\delta)}\max\{1,(\sigma_{\max}^2+\omega^2)^{-1}\}}{(1-\vartheta)^2\sqrt{Tpq_u}} \leq c \tag{6.23}$$

Under these hypotheses, we just proved that with probability at least $1-8\delta$,

$$|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| \leq c\frac{\sigma_{\max}^2+\omega^2}{(1-\vartheta)^2}\frac{\sqrt{\log(1/\delta)}}{\sqrt{Tpq_u}}.$$

We finish with a union bound, applying the previous result to all pairs $(d_1,d_2) \in [D]^2$. With probability greater than $1-8D^2\delta$, we have:

$$\max_{d_1,d_2}|(\widehat{\Gamma}_h - \Gamma_h)_{d_1,d_2}| = \|\widehat{\Gamma}_h - \Gamma_h\|_{\max} \leq c\frac{\sigma_{\max}^2+\omega^2}{(1-\vartheta)^2}\frac{\sqrt{\log(1/\delta)}}{\sqrt{Tpq_u}}.$$

Replacing δ with $8D^2\delta$ gives us the result we wanted: with probability greater than $1-\delta$,

$$\|\widehat{\Gamma}_h - \Gamma_h\|_{\max} \leq c\frac{\sigma_{\max}^2+\omega^2}{(1-\vartheta)^2}\frac{\sqrt{\log(D/\delta)}}{\sqrt{Tpq_u}}.$$

□

6.A.7 Behavior of the Dantzig selector

We now walk the final steps from the error on $\widehat{\Gamma}_h$ to the error on $\widehat{\theta}$. In order to recover Theorem 6.4.1, we adapt the convergence proof from Han, Lu, and Liu (2015, Appendix A.1). However, we use our own notations and our custom concentration results for $\widehat{\Gamma}_h$. To make comparison between both papers easier, we provide a dictionary of the main notations in Table 6.1.

Our sparse transition estimator is defined as a solution to (6.8). The end goal is to control the error $\|\widehat{\theta}-\theta\|_1$, where $\theta = \Gamma_1\Gamma_0^{-1}$ is the true transition matrix. We start by choosing a specific λ such that θ is feasible with high probability.

	This paper	Han, Lu, and Liu (2015)
VAR def	$X_t = \theta X_{t-1} + \varepsilon_t$	$X_t = A_1^\top X_{t-1} + Z_t$
Covariance	$\Gamma_h = \mathrm{Cov}(X_h, X_0)$	$\Sigma_i = \mathrm{Cov}(X_0, X_i)$
Yule-Walker	$\Gamma_h = \theta^h \Gamma_0$	$\Sigma_i = \Sigma_0 A_1^i$
Covariance estimate	$\widehat{\Gamma}_h$	S_i
Covariance error	$\mathrm{err}(\delta)$	ζ_i
Optimization constraint	$\|M\widehat{\Gamma}_0 - \widehat{\Gamma}_1\|_{\max} \le \lambda$	$\|S_0 M - S_1\|_{\max} \le \lambda$
Optimization objective	$\|\mathrm{vec}(M)\|_1$	$\|\mathrm{vec}(M)\|_1$
Threshold in proof	ν	λ_1

Table 6.1: Notation correspondence between this paper and Han, Lu, and Liu (2015)

Lemma 6.A.15 (Feasibility of the real θ)**.** *If we select the penalization level*

$$\lambda = (\|\theta\|_\infty + 1)\,\mathrm{err}(\delta),$$

then with probability at least $1-\delta$, the real θ is a feasible solution to the optimization problem (6.8).

Proof.

$$\begin{aligned}
\|\theta\widehat{\Gamma}_0 - \widehat{\Gamma}_1\|_{\max} &= \|\Gamma_1\Gamma_0^{-1}\,\widehat{\Gamma}_0 - \widehat{\Gamma}_1\|_{\max} \\
&= \|\Gamma_1\Gamma_0^{-1}\,\widehat{\Gamma}_0 - \Gamma_1 + \Gamma_1 - \widehat{\Gamma}_1\|_{\max} \\
&\le \|\Gamma_1\Gamma_0^{-1}\,\widehat{\Gamma}_0 - \Gamma_1\Gamma_0^{-1}\Gamma_0\|_{\max} + \|\Gamma_1 - \widehat{\Gamma}_1\|_{\max} \\
&= \|\theta(\widehat{\Gamma}_0 - \Gamma_0)\|_{\max} + \|\Gamma_1 - \widehat{\Gamma}_1\|_{\max}
\end{aligned}$$

By Lemma A.1.5,

$$\|\theta(\widehat{\Gamma}_0 - \Gamma_0)\|_{\max} \le \|\theta\|_\infty \|\widehat{\Gamma}_0 - \Gamma_0\|_{\max}$$

By Lemma 6.A.14, with probability greater than $1-2\delta$,

$$\|\widehat{\Gamma}_0 - \Gamma_0\|_{\max} \le \mathrm{err}(\delta) \quad \text{and} \quad \|\widehat{\Gamma}_1 - \Gamma_1\|_{\max} \le \mathrm{err}(\delta)$$

which implies

$$\|\theta\widehat{\Gamma}_0 - \widehat{\Gamma}_1\|_{\max} \le (\|\theta\|_\infty + 1)\,\mathrm{err}(\delta).$$

This is exactly the feasibility criterion for (6.8) if $\lambda = (\|\theta\|_\infty + 1)\,\mathrm{err}(\delta)$. □

Lemma 6.A.16 (Error on $\widehat{\theta}$ in max norm)**.** *If we select $\lambda = (\|\theta\|_\infty + 1)\,\mathrm{err}(\delta)$, then with probability at least $1-\delta$, the max norm error of $\widehat{\theta}$ satisfies*

$$\|\widehat{\theta} - \theta\|_{\max} \le 2\lambda\|\Gamma_0^{-1}\|_1.$$

Proof.

$$\begin{aligned}
\|\widehat{\theta} - \theta\|_{\max} &= \|\widehat{\theta} - \Gamma_1\Gamma_0^{-1}\|_{\max} \\
&= \|(\widehat{\theta}\Gamma_0 - \Gamma_1)\Gamma_0^{-1}\|_{\max} \\
&= \|(\widehat{\theta}\Gamma_0 - \widehat{\theta}\widehat{\Gamma}_0 + \widehat{\theta}\widehat{\Gamma}_0 - \widehat{\Gamma}_1 + \widehat{\Gamma}_1 - \Gamma_1)\Gamma_0^{-1}\|_{\max} \\
&\le \|(\widehat{\theta}\Gamma_0 - \widehat{\theta}\widehat{\Gamma}_0)\Gamma_0^{-1}\|_{\max} + \|(\widehat{\theta}\widehat{\Gamma}_0 - \widehat{\Gamma}_1)\Gamma_0^{-1}\|_{\max} + \|(\widehat{\Gamma}_1 - \Gamma_1)\Gamma_0^{-1}\|_{\max}
\end{aligned}$$

By Lemma A.1.5,

$$\begin{aligned}\|\widehat{\theta}-\theta\|_{\max} &\leq \left(\|\widehat{\theta}(\Gamma_0-\widehat{\Gamma}_0)\|_{\max}+\|\widehat{\theta}\widehat{\Gamma}_0-\widehat{\Gamma}_1\|_{\max}+\|\widehat{\Gamma}_1-\Gamma_1\|_{\max}\right)\|\Gamma_0^{-1}\|_1\\ &\leq \left(\|\widehat{\theta}\|_\infty\|\Gamma_0-\widehat{\Gamma}_0\|_{\max}+\|\widehat{\theta}\widehat{\Gamma}_0-\widehat{\Gamma}_1\|_{\max}+\|\widehat{\Gamma}_1-\Gamma_1\|_{\max}\right)\|\Gamma_0^{-1}\|_1\end{aligned}$$

We want to control $\|\widehat{\theta}\|_\infty$ using $\|\theta\|_\infty$. Let us recall that the operator ℓ_∞ norm is equal to the maximum ℓ_1 norm of the rows of a matrix. To control the rows of $\widehat{\theta}$, we notice that the optimization problem defining $\widehat{\theta}$, namely

$$\min_{M\in\mathbb{R}^{D\times D}}\|\mathrm{vec}(M)\|_1 \quad \text{s.t.} \quad \|M\widehat{\Gamma}_0-\widehat{\Gamma}_1\|_{\max}\leq\lambda$$

is equivalent to the row-wise minimization

$$\forall i, \quad \min_{M_{i,\cdot}\in\mathbb{R}^{1\times D}}\|M_{i,\cdot}\|_1 \quad \text{s.t.} \quad \|M_{i,\cdot}\widehat{\Gamma}_0-(\widehat{\Gamma}_1)_{i,\cdot}\|_{\max}\leq\lambda$$

From this, we deduce that each row of the optimum $\widehat{\theta}$ satisfies $\|\widehat{\theta}_{i,\cdot}\|_1\leq\|\theta_{i,\cdot}\|_1$, which implies $\|\widehat{\theta}\|_\infty\leq\|\theta\|_\infty$. Going back to our error estimate, we get:

$$\|\widehat{\theta}-\theta\|_{\max}\leq\left(\|\theta\|_\infty\|\Gamma_0-\widehat{\Gamma}_0\|_{\max}+\|\widehat{\theta}\widehat{\Gamma}_0-\widehat{\Gamma}_1\|_{\max}+\|\widehat{\Gamma}_1-\Gamma_1\|_{\max}\right)\|\Gamma_0^{-1}\|_1$$

Note that the middle term is smaller than λ because the optimum $\widehat{\theta}$ is a feasible solution. Meanwhile, the first and third term are smaller than $\mathrm{err}(\delta)$ with probability $1-\delta$:

$$\|\widehat{\theta}-\theta\|_{\max}\leq(\|\theta\|_\infty\,\mathrm{err}(\delta)+\lambda+\mathrm{err}(\delta))\,\|\Gamma_0^{-1}\|_1=2\lambda\|\Gamma_0^{-1}\|_1$$

□

To complete the proof of Theorem 6.4.1, we simply need to go from the max norm to the ℓ_∞ operator norm.

Proof. Let $\nu>0$ be a threshold (to be chosen later). We define

$$s_1=\max_i\sum_j\min\left\{\frac{|\theta_{i,j}|}{\nu},1\right\} \quad \text{and} \quad \mathcal{I}_i=\{j:|\theta_{i,j}|\geq\nu\}$$

With high probability, the following holds for any row i:

$$\begin{aligned}\|\widehat{\theta}_{i,\cdot}-\theta_{i,\cdot}\|_1 &\leq \|\widehat{\theta}_{i,\mathcal{I}_i^c}-\theta_{i,\mathcal{I}_i^c}\|_1+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &\leq \|\widehat{\theta}_{i,\mathcal{I}_i^c}\|_1+\|\theta_{i,\mathcal{I}_i^c}\|_1+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &= (\|\widehat{\theta}_{i,\cdot}\|_1-\|\widehat{\theta}_{i,\mathcal{I}_i}\|_1)+\|\theta_{i,\mathcal{I}_i^c}\|_1+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &\leq \|\theta_{i,\cdot}\|_1-\|\widehat{\theta}_{i,\mathcal{I}_i}\|_1+\|\theta_{i,\mathcal{I}_i^c}\|_1+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &= (\|\theta_{i,\mathcal{I}_i}\|_1+\|\theta_{i,\mathcal{I}_i^c}\|_1)-\|\widehat{\theta}_{i,\mathcal{I}_i}\|_1+\|\theta_{i,\mathcal{I}_i^c}\|_1+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &= 2\|\theta_{i,\mathcal{I}_i^c}\|_1+(\|\theta_{i,\mathcal{I}_i}\|_1-\|\widehat{\theta}_{i,\mathcal{I}_i}\|_1)+\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\\ &\leq 2\|\theta_{i,\mathcal{I}_i^c}\|_1+2\|\widehat{\theta}_{i,\mathcal{I}_i}-\theta_{i,\mathcal{I}_i}\|_1\end{aligned}$$

By definition of $\mathcal{I}_i$, for all $j \in \mathcal{I}_i^c$, $|\theta_{i,j}| \leq \nu$, hence

$$\|\theta_{i,\mathcal{I}_i^c}\|_1 = \sum_{j \in \mathcal{I}_i^c} |\theta_{i,j}| = \sum_{j \in \mathcal{I}_i^c} \min\{|\theta_{i,j}|, \nu\} \leq \sum_j \min\{|\theta_{i,j}|, \nu\} \leq \nu s_1$$

Meanwhile, the second term satisfies

$$\|\widehat{\theta}_{i,\mathcal{I}_i} - \theta_{i,\mathcal{I}_i}\|_1 \leq |\mathcal{I}_i| \times \|\widehat{\theta} - \theta\|_{\max}$$

And by definition of $\mathcal{I}_i$, for all $j \in \mathcal{I}_i$, $|\theta_{i,j}| \geq \nu$, hence

$$|\mathcal{I}_i| = \sum_{j \in \mathcal{I}_i} 1 = \sum_{j \in \mathcal{I}_i} \min\left\{\frac{|\theta_{i,j}|}{\nu}, 1\right\} \leq \sum_j \min\left\{\frac{|\theta_{i,j}|}{\nu}, 1\right\} \leq s_1$$

Combining all of this, we get that with high probability,

$$\|\widehat{\theta}_{i,\cdot} - \theta_{i,\cdot}\|_1 \leq 2(\nu + 2\lambda\|\Gamma_0^{-1}\|_1)s_1$$

Judging by the last Equation, it makes sense to choose $\nu = 2\lambda\|\Gamma_0^{-1}\|_1$. Furthermore, our sparsity hypothesis on θ implies that for all but s of the coefficients of any row i, $\min\{|\theta_{i,j}|, \nu\} = |\theta_{i,j}| = 0$. We deduce that for every i,

$$\sum_j \min\left\{|\theta_{i,j}|, \nu\right\} \leq s \max_j \min\left\{|\theta_{i,j}|, \nu\right\} \leq \nu s$$

which directly implies

$$\nu s_1 = \max_i \sum_j \min\left\{|\theta_{i,j}|, \nu\right\} \leq \nu s$$

We finally find that with high probability,

$$\|\widehat{\theta}_{i,\cdot} - \theta_{i,\cdot}\|_1 \leq 4\nu s_1 \leq 4\nu s = 8\lambda\|\Gamma_0^{-1}\|_1 s$$

With the help of a union bound, again with high probability,

$$\|\widehat{\theta} - \theta\|_\infty = \max_i \|\widehat{\theta}_{i,\cdot} - \theta_{i,\cdot}\|_1 \leq 8\lambda\|\Gamma_0^{-1}\|_1 s$$

We substitute the value of λ and obtain

$$\|\widehat{\theta} - \theta\|_\infty \leq 8(\|\theta\|_\infty + 1)\,\mathrm{err}(\delta)\|\Gamma_0^{-1}\|_1 s$$

Once we plug in the value of $\mathrm{err}(\delta)$, the resulting high-probability error bound reads

$$\|\widehat{\theta} - \theta\|_\infty \leq c\frac{\|\theta\|_\infty + 1}{\|\Gamma_0^{-1}\|_1^{-1}} \frac{\sigma_{\max}^2 + \omega^2}{(1-\vartheta)^2} \frac{s\sqrt{\log(D/\delta)}}{\sqrt{Tpq_u}}$$

Since ϑ only acted as an upper bound on $\|\theta\|_2$ in this proof, we can define

$$\gamma_u(\theta) = \frac{\|\theta\|_\infty + 1}{(1 - \|\theta\|_2)^2} \frac{\sigma_{\max}^2 + \omega^2}{\|\Gamma_0^{-1}\|_1^{-1}}$$

to obtain the compressed expression

$$\|\widehat{\theta} - \theta\|_\infty \leq c\gamma_u(\theta)\frac{s\sqrt{\log(D/\delta)}}{\sqrt{Tpq_u}}.$$

□

6.B Proof of the minimax lower bound

We now present the detailed proof of Theorem 6.4.2.

6.B.1 Overview

Our argument is based on Fano's method, which we sum up in Lemma A.2.1. For a detailed presentation, we refer the reader to Tsybakov (2009, Chapter 2). Note that Wainwright (2019, Chapter 15) and Duchi (2019, Chapter 7) also offer good treatments of the subject.

Fano's method relies on choosing a set of parameters $\theta_0, \theta_1, ..., \theta_M$ satisfying two seemingly contradictory conditions: their induced distributions must be hard to distinguish, yet they must lie as fart apart from one another as possible. In particular, the crucial requirement of Fano's method is a tight upper bound on the KL divergence between two distributions generated by different parameters θ_i and θ_0. Taking the latter to be 0, we actually want to bound

$$\frac{1}{M+1}\sum_{i=1}^{M} \mathrm{KL}\left\{\mathbb{P}_{\theta_i}(\Pi, Y) \parallel \mathbb{P}_0(\Pi, Y)\right\} \leq \max_i \mathrm{KL}\left\{\mathbb{P}_{\theta_i}(\Pi, Y) \parallel \mathbb{P}_0(\Pi, Y)\right\}$$

By Lemma A.2.2,

$$\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(\Pi, Y) \parallel \mathbb{P}_0(\Pi, Y)\right\} = \mathrm{KL}\left\{\mathbb{P}_{\theta_i}(\Pi) \parallel \mathbb{P}_0(\Pi)\right\} + \mathbb{E}_\Pi\left[\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right]$$

Since θ_i does not affect the distribution of the sampling process Π, the first term of the right-hand side is zero, and we will concentrate on the second term. First, we will upper-bound the random variable inside the expectation for a fixed realization of Π, and then we will average said bound over all possible projections.

We now give the structure of the argument in a coherent order, along with the most important intermediate results:

1. Compute the conditional covariance $\mathrm{Cov}_\theta[Y \mid \Pi]$ and decompose it into a constant term Q_Π (corresponding to the independent case $\theta = 0$) plus a residual $R_\Pi(\theta)$ (Lemma 6.B.1).
2. Upper-bound the conditional KL divergence $\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}$ using the "deviations from the identity" $\Delta_\Pi(\theta) = Q_\Pi^{-1/2} R_\Pi(\theta) Q_\Pi^{-1/2}$ (Lemma 6.B.2).
3. Control $\Delta_\Pi(\theta)$ using features of $R(\theta)$ scaled by sampling-related factors (Lemmas 6.B.3, 6.B.4 and 6.B.5).
4. Deduce an upper bound on the KL divergence $\mathbb{E}_\Pi[\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}]$ (Lemma 6.B.6).
5. Apply Fano's method to a set of parameters θ_i constructed from a pruned binary hypercube of well-chosen radius.

6.B.2 Change of notations

For this part, we slightly modify the previous conventions: we now assume that all the rows of Π_t that contain only zeros are removed. In other words, Π_t is no longer the diagonal matrix $\mathrm{diag}(\pi_t)$ but instead becomes a wide rectangular matrix with exactly one 1 per row and at most one 1 per

column. We thus have $\Pi\Pi^\top = I$ unless all the $\pi_{t,d}$ are zero, in which case the matrix Π is empty, and so are the observations Y. Let us denote this very unlikely event by E, and its complement by E^c. If Π is such that E happens, we obviously have $\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\} = 0$, which means that

$$\mathbb{E}_\Pi\left[\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] = \mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] \tag{6.24}$$

For the beginning of the proof, we consider a fixed, non-empty realization of Π.

6.B.3 Covariance decomposition

As we announced in the proof sketch, our reference parameter will be $\theta_0 = 0$, which is why it makes sense to express the conditional covariance of Y as a deviation from the case without interactions. This is the aim of the following result.

Lemma 6.B.1 (Conditional covariance decomposition)**.** *The covariance matrix of Y given Π decomposes as*

$$\mathrm{Cov}_\theta[Y \mid \Pi] = Q_\Pi + R_\Pi(\theta),$$

where Q_Π is a constant term and $R_\Pi(\theta)$ is a residual which vanishes as $\theta \to 0$. They are defined as follows: the constant term is

$$Q_\Pi = \Pi(\mathrm{bdiag}_T\,\Sigma)\Pi^\top + \omega^2 I$$

whereas the residual equals

$$R_\Pi(\theta) = \Pi R(\theta)\Pi^\top \qquad \textit{with} \qquad R(\theta) = \begin{bmatrix} \theta\Gamma_0(\theta)\theta^\top & \Gamma_0(\theta)(\theta^1)^\top & \Gamma_0(\theta)(\theta^2)^\top & \cdots \\ \theta^1\Gamma_0(\theta) & \theta\Gamma_0(\theta)\theta^\top & \Gamma_0(\theta)(\theta^1)^\top & \\ \theta^2\Gamma_0(\theta) & \theta^1\Gamma_0(\theta) & \theta\Gamma_0(\theta)\theta^\top & \\ \vdots & & & \ddots \end{bmatrix}.$$

Proof. We use Equation (6.5) to see that the conditional distribution $\mathbb{P}_\theta(Y \mid \Pi)$ is a centered multivariate Gaussian with covariance

$$\mathrm{Cov}_\theta[Y \mid \Pi] = \omega^2 I + \Pi\,\mathrm{Cov}_\theta[X]\Pi^\top.$$

We then use Lemma 6.A.1 to get an expression of $\mathrm{Cov}_\theta[X]$ and deduce that its constant term (w.r.t to θ) is a block-diagonal matrix filled with copies of Σ:

$$\mathrm{Cov}_\theta[Y \mid \Pi] = \omega^2 I + \Pi\,\mathrm{bdiag}_T(\Sigma)\Pi^\top + \Pi\left(\mathrm{Cov}_\theta[X] - \mathrm{bdiag}_T(\Sigma)\right)\Pi^\top.$$

Finally, we define $Q_\Pi = \omega^2 I + \Pi\,\mathrm{bdiag}_T(\Sigma)\Pi^\top$, $R(\theta) = \mathrm{Cov}_\theta[X] - \mathrm{bdiag}_T(\Sigma)$ and $R_\Pi(\theta) = \Pi R(\theta)\Pi^\top$ to obtain the decomposition we announced. The diagonal blocks of $R(\theta)$ are easily computed by noticing that $\Gamma_0(\theta) - \Sigma = \theta\Gamma_0(\theta)\theta^\top$. □

6.B.4 From the KL divergence to $\Delta_\Pi(\theta)$

Judging by Lemma 6.B.1, choosing a parameter θ close to 0 yields a conditional distribution for Y whose covariance is close to Q_Π. In the next result, we translate this into a bound on the KL divergence between $\mathbb{P}_\theta(Y \mid \Pi)$ and $\mathbb{P}_0(Y \mid \Pi)$.

Lemma 6.B.2. *Let us define the deviation from the identity:*

$$\Delta_\Pi(\theta) = Q_\Pi^{-1/2} R_\Pi(\theta) Q_\Pi^{-1/2}.$$

Then the conditional KL divergence is upper-bounded by:

$$\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\} \leq \frac{\|\Delta_\Pi(\theta)\|_F^2}{2(1+\lambda_{\min}(\Delta_\Pi(\theta)))}.$$

Proof. The conditional KL divergence $\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}$ can be bounded with the help of Lemma A.2.4. Indeed, both conditional distributions are Gaussian and have the same expectation, and covariance matrices that are "close" in the following sense: by Lemma 6.B.1,

$$\begin{aligned}
\mathrm{Cov}_0(Y \mid \Pi) &= Q_\Pi = Q_\Pi^{1/2}(Q_\Pi^{1/2})^\top \\
\mathrm{Cov}_\theta(Y \mid \Pi) &= Q_\Pi + R_\Pi(\theta) = Q_\Pi^{1/2}\Big(I + \underbrace{Q_\Pi^{-1/2} R_\Pi(\theta) Q_\Pi^{-1/2}}_{\Delta_\Pi(\theta)}\Big)(Q_\Pi^{1/2})^\top.
\end{aligned}$$

By Lemma A.1.2, there exists a real number $r_{\min} \geq \varsigma_{\min}\left(Q_\Pi^{1/2}\right)^2 = \varsigma_{\min}(Q_\Pi)$ such that

$$\lambda_{\min}(\mathrm{Cov}_\theta(Y \mid \Pi)) = r_{\min}\lambda_{\min}(I + \Delta_\Pi(\theta)).$$

Since $Q_\Pi \succeq \omega^2 I \succ 0$, its minimum singular value satisfies $\varsigma_{\min}(Q_\Pi) > 0$, so that $r_{\min} > 0$. In addition, $\mathrm{Cov}_\theta(Y \mid \Pi) \succeq \omega^2 I \succ 0$, so that $\lambda_{\min}(\mathrm{Cov}_\theta(Y \mid \Pi)) > 0$. Therefore,

$$\lambda_{\min}(I + \Delta_\Pi(\theta)) = \frac{\lambda_{\min}(\mathrm{Cov}_\theta(Y \mid \Pi))}{r_{\min}} > 0 \quad \text{and} \quad \lambda_{\min}(\Delta_\Pi(\theta)) > -1,$$

which means we can apply Lemma A.2.4 with $\mathbb{P}_1 = \mathbb{P}_\theta(Y \mid \Pi)$ and $\mathbb{P}_0 = \mathbb{P}_0(Y \mid \Pi)$. □

6.B.5 From $\Delta_\Pi(\theta)$ to $R_\Pi(\theta)$

Lemma 6.B.2 strongly suggests studying a certain fraction involving $\Delta_\Pi(\theta)$. In the following result, we boil it down to a function of the residual term $R_\Pi(\theta)$.

Lemma 6.B.3. *Assume $\|R(\theta)\|_2 \leq (\sigma_{\min}^2 + \omega^2)/2$. We have the following upper bound:*

$$\frac{\|\Delta_\Pi(\theta)\|_F^2}{2(1+\lambda_{\min}(\Delta_\Pi(\theta)))} \leq \frac{\|R_\Pi(\theta)\|_F^2}{\left(\sigma_{\min}^2 + \omega^2\right)^2}.$$

Proof. Since the quantity $\lambda_{\min}(\Delta_\Pi(\theta))$ in the denominator is hard to control, we will work with the spectral norm instead. Indeed, whenever $\|\Delta_\Pi(\theta)\|_2 < 1$, we have the crude bound

$$\frac{1}{1-\lambda_{\min}(\Delta_\Pi(\theta))} \leq \frac{1}{1-\|\Delta_\Pi(\theta)\|_2}.$$

Let us start by noticing that, thanks to Lemma A.1.4,

$$\begin{aligned}
\|\Delta_\Pi(\theta)\|_F^2 &= \|Q_\Pi^{-1/2} R_\Pi(\theta) Q_\Pi^{-1/2}\|_F^2 \leq \|Q_\Pi^{-1/2}\|_2^4 \|R_\Pi(\theta)\|_F^2 = \|Q_\Pi^{-1}\|_2^2 \|R_\Pi(\theta)\|_F^2 \\
\|\Delta_\Pi(\theta)\|_2 &= \|Q_\Pi^{-1/2}\Pi R(\theta)\Pi^\top Q_\Pi^{-1/2}\|_2 \leq \|Q_\Pi^{-1/2}\Pi\|_2^2 \|R(\theta)\|_2.
\end{aligned}$$

We will later see how the spectral and Frobenius norms of the residual $R(\theta)$ can be controlled as a function of θ. For now, we must work to upper bound $\|Q_\Pi^{-1}\|_2$ and $\|Q_\Pi^{-1}\Pi\|_2^2$.

To simplify the following proof, we write $\Sigma_\mathrm{d} = \mathrm{bdiag}_T\, \Sigma$. Since Σ_d is block-diagonal, its spectrum is the same as the spectrum of Σ repeated T times, hence $\lambda_{\min}(\Sigma_\mathrm{d}) = \sigma_{\min}^2$. And since we assumed E^c happens (non-empty projection), we have $\Pi\Pi^\top = I$ and $\Pi^\top\Pi = \mathrm{diag}(\pi)$, which has at least one entry equal to 1.

We start with $\|Q_\Pi^{-1}\|_2$. Since $Q_\Pi \succeq \omega^2 I \succ 0$ is non-singular and symmetric,

$$\|Q_\Pi^{-1}\|_2 = \lambda_{\max}(Q_\Pi^{-1}) = \frac{1}{\lambda_{\min}(Q_\Pi)} = \frac{1}{\lambda_{\min}(\Pi\Sigma_\mathrm{d}\Pi^\top + \omega^2 I)}.$$

Remembering that $\Sigma_\mathrm{d} \succeq \sigma_{\min}^2 I$, we get

$$\Pi\Sigma_\mathrm{d}\Pi^\top + \omega^2 I \succeq \sigma_{\min}^2 \Pi\Pi^\top + \omega^2 I = (\sigma_{\min}^2 + \omega^2) I$$

and thus

$$\|Q_\Pi^{-1}\|_2 \leq \frac{1}{\sigma_{\min}^2 + \omega^2}.$$

We now continue with $\|Q_\Pi^{-1/2}\Pi\|_2^2$. By definition of the spectral norm,

$$\|Q_\Pi^{-1/2}\Pi\|_2^2 = \lambda_{\max}\left(\Pi^\top Q_\Pi^{-1}\Pi\right) = \lambda_{\max}\left(\Pi^\top(\Pi\Sigma_\mathrm{d}\Pi^\top + \omega^2 I)^{-1}\Pi\right).$$

Because matrix inversion is decreasing w.r.t. the Loewner order on positive semi-definite matrices,

$$(\Pi\Sigma_\mathrm{d}\Pi^\top + \omega^2 I)^{-1} \preceq (\sigma_{\min}^2 + \omega^2)^{-1} I^{-1}$$
$$\Pi^\top(\Pi\Sigma_\mathrm{d}\Pi^\top + \omega^2 I)^{-1}\Pi \preceq \frac{1}{\sigma_{\min}^2 + \omega^2}\Pi^\top\Pi.$$

It follows that

$$\|Q_\Pi^{-1/2}\Pi\|_2^2 \leq \frac{1}{\sigma_{\min}^2 + \omega^2}\lambda_{\max}(\Pi^\top\Pi) = \frac{1}{\sigma_{\min}^2 + \omega^2}.$$

The conclusion is within reach:

$$\begin{aligned}\frac{\|\Delta_\Pi(\theta)\|_F^2}{1 + \lambda_{\min}(\Delta_\Pi(\theta))} &\leq \frac{\|\Delta_\Pi(\theta)\|_F^2}{1 - \|\Delta_\Pi(\theta)\|_2} \leq \frac{\|Q_\Pi^{-1}\|_2^2\|R_\Pi(\theta)\|_F^2}{1 - \|Q_\Pi^{-1/2}\Pi\|_2^2\|R(\theta)\|_2} \\ &\leq \frac{\left(\frac{1}{\sigma_{\min}^2+\omega^2}\right)^2\|R_\Pi(\theta)\|_F^2}{1 - \frac{1}{\sigma_{\min}^2+\omega^2}\|R(\theta)\|_2} \leq \frac{2\|R_\Pi(\theta)\|_F^2}{(\sigma_{\min}^2 + \omega^2)^2}\end{aligned}$$

The last inequality is justified by our assumption $\|R(\theta)\|_2 \leq (\sigma_{\min}^2 + \omega^2)/2$. Another consequence of this assumption is that

$$\|\Delta_\Pi(\theta)\|_2 \leq \|Q_\Pi^{-1/2}\Pi\|_2^2\|R(\theta)\|_2 \leq \frac{1}{\sigma_{\min}^2 + \omega^2}\frac{\sigma_{\min}^2 + \omega^2}{2} = \frac{1}{2} < 1$$

which is sufficient for the first inequality to hold. □

6.B.6 From $R_\Pi(\theta)$ to $R(\theta)$

As the previous Lemma underlines, the last step we need to get rid of the dependency in Π is to study the average norm of $R_\Pi(\theta)$.

Lemma 6.B.4. *Let* $q_\ell = \max\{1-b, 2p-(1-b)\}$. *Then*

$$\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\|R_\Pi(\theta)\|_F^2\right] \le p\,\mathrm{Tr}[R(\theta)\odot R(\theta)] + pq_\ell\|R(\theta)\|_F^2.$$

Proof. We first notice that for any matrix A,

$$\begin{aligned}\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\|\Pi A\Pi^\top\|_F^2\right] &= \mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\,\mathrm{Tr}\left[\Pi A\Pi^\top\Pi A^\top\Pi^\top\right]\right]\\ &= \mathbb{E}_\Pi\left[\mathrm{Tr}\left[\,\mathrm{diag}(\pi)A\,\mathrm{diag}(\pi)A^\top\right]\right]\\ &= \sum_{i,j}\mathbb{E}_\Pi[\pi_i\pi_j]A_{i,j}^2.\end{aligned}$$

We can apply this to $R_\Pi(\theta) = \Pi R(\theta)\Pi^\top$:

$$\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\|R_\Pi(\theta)\|_F^2\right] = \sum_{i,j}\mathbb{E}_\Pi[\pi_i\pi_j]R(\theta)_{i,j}^2.$$

The rest of the proof consists in plugging in the moments $\mathbb{E}_\Pi[\pi_i\pi_j]$ from Lemma 6.A.4:

$$\begin{aligned}\mathbb{E}_\Pi\left[\|R_\Pi(\theta)\|_F^2\right] = &\sum_{\substack{t_1,t_2,d_1,d_2\\(t_1,d_1)=(t_2,d_2)}} pR(\theta)^2_{(t_1,d_1),(t_2,d_2)}\\ &+\sum_{\substack{t_1,t_2,d_1,d_2\\d_1\neq d_2}} p^2R(\theta)^2_{(t_1,d_1),(t_2,d_2)}\\ &+\sum_{\substack{t_1,t_2,d_1,d_2\\d_1=d_2,t_1\neq t_2}} (p^2+p(1-p)(1-a-b)^{|t_1-t_2|})R(\theta)^2_{(t_1,d_1),(t_2,d_2)}.\end{aligned}$$

The sum in the last term can be crudely controlled as follows:

$$\begin{aligned}\sum_{\substack{t_1,t_2,d\\t_1\neq t_2}}(1-a-b)^{|t_1-t_2|}R(\theta)^2_{(t_1,d),(t_2,d)} &\le |1-a-b|\sum_{\substack{t_1,t_2\\t_1\neq t_2}}\sum_d (R(\theta)_{[t_1,t_2]})^2_{d,d}\\ &\le |1-a-b|\sum_{t_1\neq t_2}\|R(\theta)_{[t_1,t_2]}\|_F^2\\ &\le |1-a-b|\cdot\|R(\theta)\|_F^2\end{aligned}$$

This yields a short, but probably suboptimal bound:

$$\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\|R_\Pi(\theta)\|_F^2\right] \le p\,\mathrm{Tr}[R(\theta)\odot R(\theta)] + (p^2+p(1-p)|1-a-b|)\|R(\theta)\|_F^2.$$

In the previous part, we already saw that

$$p+(1-p)(1-a-b) = 1-b.$$

Similarly, we obtain

$$\begin{aligned} p + (1-p)(a+b-1) &= \frac{a}{a+b} + \frac{b}{a+b}(a+b-1) \\ &= \frac{a+ba+b^2-b}{a+b} = \frac{a(1+b)-b(1-b)}{a+b} \\ &= p(1+b) - (1-p)(1-b) = 2p - (1-b). \end{aligned}$$

As a consequence,

$$p + (1-p)|1-a-b| = \max\{1-b, 2p-(1-b)\} = q_\ell,$$

which yields the expected result. □

6.B.7 Bounding $R(\theta)$

Lemma 6.B.4 relates the bounds involving $R_\Pi(\theta)$ to features of the full residual $R(\theta)$, which we now study.

Lemma 6.B.5. *The residual $R(\theta)$ satisfies the following inequalities:*

$$\begin{aligned} \|R(\theta)\|_2 &\leq \frac{2\sigma_{\max}^2}{(1-\vartheta)^2}\|\theta\|_2 \\ \|R(\theta)\|_F^2 &\leq \frac{2T\sigma_{\max}^4}{(1-\vartheta)^3}\|\theta\|_F^2 \\ \mathrm{Tr}[R(\theta)\odot R(\theta)] &\leq \frac{T\sigma_{\max}^4}{(1-\vartheta)^2}\|\theta\|_2^2\|\theta\|_F^2. \end{aligned}$$

Proof. We start by giving a formula for the blocks of $R(\theta)$: by Lemma 6.B.1,

$$R(\theta)_{[t,s]} = \begin{cases} \theta^{t-s}\Gamma_0(\theta) & \text{if } s \in [1, t-1] \\ \theta\Gamma_0(\theta)\theta^\top & \text{if } s = t \\ \Gamma_0(\theta)(\theta^{t-s})^\top & \text{if } s \in [t+1, T]. \end{cases}$$

These individual blocks can be bounded using Lemmas A.1.4 and 6.A.2: if $r \geq 1$, then

$$\begin{aligned} \|\theta^r\Gamma_0(\theta)\|_F^2 &\leq \|\Gamma_0(\theta)\|_2^2\|\theta^r\|_F^2 \leq \|\Gamma_0(\theta)\|_2^2\|\theta\|_F^2\|\theta^{r-1}\|_2^2 \leq \frac{\sigma_{\max}^4}{(1-\vartheta)^2}\|\theta\|_F^2\|\theta\|_2^{2(r-1)} \\ \|\Gamma_0(\theta)(\theta^r)^\top\|_F^2 &\leq \|\Gamma_0(\theta)\|_2^2\|\theta^r\|_F^2 \leq \|\Gamma_0(\theta)\|_2^2\|\theta\|_F^2\|\theta^{r-1}\|_2^2 \leq \frac{\sigma_{\max}^4}{(1-\vartheta)^2}\|\theta\|_F^2\|\theta\|_2^{2(r-1)} \\ \|\theta\Gamma_0(\theta)\theta^\top\|_F^2 &\leq \|\theta\|_2^2\|\Gamma_0(\theta)\|_2^2\|\theta\|_F^2 \leq \frac{\sigma_{\max}^4}{(1-\vartheta)^2}\|\theta\|_F^2\|\theta\|_2^2. \end{aligned}$$

Since we control the norm of each block of $R(\theta)$, we control the norm of the whole matrix:

$$\begin{aligned}
\|R(\theta)\|_F^2 &= \sum_{t=1}^{T}\left(\sum_{s=1}^{t-1}\|\theta^{t-s}\Gamma_0(\theta)\|_F^2 + \|\theta\Gamma_0(\theta)\theta\|_F^2 + \sum_{s=t+1}^{T}\|\Gamma_0(\theta)\theta^{s-t}\|_F^2\right)\\
&\leq \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}\sum_{t=1}^{T}\left(\sum_{s=1}^{t-1}\|\theta\|_2^{2(t-s-1)} + \|\theta\|_2^2 + \sum_{s=t+1}^{T}\|\theta\|_2^{2(s-t-1)}\right)\\
&\leq \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}\sum_{t=1}^{T}\left(\sum_{s=-\infty}^{t-1}\|\theta\|_2^{2(t-1-s)} + \|\theta\|_2^2 + \sum_{s=t+1}^{+\infty}\|\theta\|_2^{2(s-1-t)}\right)\\
&= \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}T\left(\frac{1}{1-\|\theta\|_2^2} + \|\theta\|_2^2 + \frac{1}{1-\|\theta\|_2^2}\right)
\end{aligned}$$

We now remember our hypothesis $\|\theta\|_2 \leq \vartheta < 1$:

$$\begin{aligned}
\|R(\theta)\|_F^2 &\leq \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}T\left(\frac{1}{1-\vartheta^2} + \vartheta^2 + \frac{1}{1-\vartheta^2}\right)\\
&= \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}T\left(\frac{2+\vartheta^2(1-\vartheta^2)}{1-\vartheta^2}\right)\\
&\leq \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}T\left(\frac{2+2\vartheta}{1-\vartheta^2}\right) = \frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta^2)^2}T\left(\frac{2}{1-\vartheta}\right)\\
&= 2T\frac{\sigma_{\max}^4\|\theta\|_F^2}{(1-\vartheta)^3}.
\end{aligned}$$

Now that we have a handle on the Frobenius norm of $R(\theta)$, we move on to its spectral norm. Notice that $R(\theta)$ can be written as a sum of Kronecker products with the subdiagonal matrices J_t:

$$R(\theta) = I \otimes \theta\Gamma_0(\theta)\theta^\top + \sum_{t=1}^{T-1}\left[J_t \otimes \theta^t\Gamma_0(\theta) + J_t^\top \otimes \Gamma_0(\theta)(\theta^t)^\top\right].$$

We can use Lemma A.1.3 and write:

$$\begin{aligned}
\|R(\theta)\|_2 &\leq \|I\|_2 \times \|\theta\Gamma_0(\theta)\theta^\top\|_2 + \sum_{t=1}^{T-1}\left[\|J_t\|_2 \times \|\theta^t\Gamma_0(\theta)\|_2 + \|J_t^\top\|_2 \times \|\Gamma_0(\theta)(\theta^t)^\top\|_2\right]\\
&\leq \|\Gamma_0(\theta)\|_2\left(\|\theta\|_2^2 + 2\sum_{t=1}^{T-1}\|\theta\|_2^t\right) \leq \frac{\sigma_{\max}^2}{1-\vartheta^2}\left(\|\theta\|_2^2 + 2\frac{\|\theta\|_2}{1-\|\theta\|_2}\right)\\
&\leq \frac{\sigma_{\max}^2\|\theta\|_2}{1-\vartheta^2}\left(\vartheta + \frac{2}{1-\vartheta}\right) \leq \frac{\sigma_{\max}^2\|\theta\|_2}{1-\vartheta}\left(\frac{2+2\vartheta}{1-\vartheta^2}\right)\\
&= 2\frac{\sigma_{\max}^2\|\theta\|_2}{(1-\vartheta)^2}.
\end{aligned}$$

We finish with the trace of the Hadamard product $R(\theta) \odot R(\theta)$.

$$\begin{aligned}
\operatorname{Tr}[R(\theta) \odot R(\theta)] &= T\operatorname{Tr}[(\theta\Gamma_0(\theta)\theta^\top) \odot (\theta\Gamma_0(\theta)\theta^\top)]\\
&\leq T\|\theta\Gamma_0(\theta)\theta^\top\|_F^2 \leq T\sigma_{\max}^4\frac{\|\theta\|_2^2\|\theta\|_F^2}{(1-\vartheta)^2}.
\end{aligned}$$

□

6.B.8 Upper bound on the KL divergence

We now have all the tools in hand to extract a KL divergence bound.

Lemma 6.B.6 (Final KL bound)**.** *Assume $\theta \in \Theta_s$ satisfies*

$$\|\theta\|_2 \leq \frac{(1-\vartheta)^2(\sigma_{\min}^2+\omega^2)}{4\sigma_{\max}^2}$$

then the expected conditional KL divergence is upper-bounded as follows:

$$\mathbb{E}_\Pi\left[\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] \leq \mathrm{KL}_{\mathrm{avg}}(\|\theta\|_2, \|\theta\|_F)$$

where we defined

$$\gamma_\ell = (1-\vartheta)^{3/2}\frac{\sigma_{\min}^2+\omega^2}{\sigma_{\max}^2} \qquad \textit{and} \qquad \mathrm{KL}_{\mathrm{avg}}(\|\theta\|_2, \|\theta\|_F) = \frac{2Tp(\|\theta\|_2^2+q_\ell)\|\theta\|_F^2}{\gamma_\ell}.$$

Proof. Let us start with Lemma 6.B.2 on the conditional KL divergence between $\mathbb{P}_\theta(Y \mid \Pi)$ and $\mathbb{P}_0(Y \mid \Pi)$: for any non-empty Π,

$$\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\} \leq \frac{\|\Delta_\Pi(\theta)\|_F^2}{2(1+\lambda_{\min}(\Delta_\Pi(\theta)))}$$

We continue with Lemma 6.B.3 linking $\Delta_\Pi(\theta)$ to $R_\Pi(\theta)$. As long as $\|R(\theta)\|_2 \leq (\sigma_{\min}^2+\omega^2)/2$ (we will see to that at the end), we have

$$\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\} \leq \frac{\|R_\Pi(\theta)\|_F^2}{(\sigma_{\min}^2+\omega^2)^2}.$$

Taking the expectation on the event E^c yields:

$$\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] \leq \frac{\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\|R_\Pi(\theta)\|_F^2\right]}{(\sigma_{\min}^2+\omega^2)^2}$$

We can now apply Lemma 6.B.4:

$$\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] \leq \frac{p\,\mathrm{Tr}[R(\theta)\odot R(\theta)] + pq_\ell\|R(\theta)\|_F^2}{(\sigma_{\min}^2+\omega^2)^2}$$

We substitute the residual bounds from Lemma 6.B.5:

$$\begin{aligned}
\mathbb{E}_\Pi\left[\mathbb{1}_{E^c}\mathrm{KL}\left\{\mathbb{P}_\theta(Y \mid \Pi) \parallel \mathbb{P}_0(Y \mid \Pi)\right\}\right] &\leq \frac{p \times \frac{T\sigma_{\max}^4}{(1-\vartheta)^2}\|\theta\|_2^2\|\theta\|_F^2 + pq_\ell \times \frac{2T\sigma_{\max}^4}{(1-\vartheta)^3}\|\theta\|_F^2}{(\sigma_{\min}^2+\omega^2)^2} \\
&\leq \left(\frac{\sigma_{\max}^2}{\sigma_{\min}^2+\omega^2}\right)^2 \frac{2Tp(\|\theta\|_2^2+q_\ell)\|\theta\|_F^2}{(1-\vartheta)^3} \\
&= \frac{2Tp(\|\theta\|_2^2+q_\ell)\|\theta\|_F^2}{\gamma_\ell}.
\end{aligned}$$

By Equation (6.24), this is equivalent to bounding the expected KL divergence regardless of the event E, hence the result. Note that our assumption on θ, combined with Lemma 6.B.5, implies

$$\|R(\theta)\|_2 \leq \frac{2\sigma_{\max}^2}{(1-\vartheta)^2}\|\theta\|_2 \leq \frac{2\sigma_{\max}^2}{(1-\vartheta)^2}\frac{(1-\vartheta)^2(\sigma_{\min}^2+\omega^2)}{4\sigma_{\max}^2} \leq \frac{\sigma_{\min}^2+\omega^2}{2}$$

□

6.B.9 Application of Fano's method

Given the KL bound we just obtained, we are finally able to prove Theorem 6.4.2.

Proof. Fano's method requires finding $M+1$ parameters θ_i such that $\theta_0 = 0$ and $\|\theta_i - \theta_j\|_F \geq 2\tau$ for $i \neq j$ (with τ to be specified), while keeping control upon the average KL divergence between the probability distributions $\mathbb{P}_{\theta_i}$ and $\mathbb{P}_0$. Judging by Lemma 6.B.6, one way to achieve this control on the KL divergence is to bound the $\|\theta_i\|_F$ uniformly in i (in other words, to choose them all inside a ball of fixed radius). We will then have to see how many 2τ-separated matrices we can fit in such a ball.

Let us consider the set $\mathcal{H}(r)$ of all block-diagonal $D \times D$ matrices with coefficients in $\{0, r\}$ such that each block has size $s \times s$ (we assume s divides D). In particular, these matrices are all row- and column-sparse, with no more than s non-zero coefficients per row or column. In terms of dimensionality, we are dealing with the (scaled) matrix equivalent of a Ds-dimensional hypercube, hence the notation $\mathcal{H}(r)$. It has cardinality 2^{Ds} and for every $\theta \in \mathcal{H}$, we have the following norm bounds:

$$\|\theta\|_2 \leq rs \qquad \text{and} \qquad \|\theta\|_F \leq r\sqrt{Ds}.$$

The spectral norm bound on θ is obtained as the maximum spectral norm of each block, which we in turn control using the Frobenius norm of each block.

Unfortunately, in this hypercube, not all pairs of vertices are well-separated. That is why we need the Gilbert-Varshamov bound of Lemma A.2.9: according to this result, there exists a pruned subset $\mathcal{K}(r) \subset \mathcal{H}(r)$ containing 0 and such that

$$|\mathcal{K}(r)| \geq |\mathcal{H}(r)|^{1/8} = 2^{Ds/8} \qquad \text{and} \qquad \|\mathrm{vec}(\theta_i) - \mathrm{vec}(\theta_j)\|_1 \geq \frac{rDs}{8}$$

for all pairs of distinct vertices θ_i and θ_j in $\mathcal{K}(r)$. We choose our set of parameters $\theta_0, \theta_1, ..., \theta_M$ to be exactly this pruned subset $\mathcal{K}(r)$, in particular $M+1 = |\mathcal{K}(r)|$.

The missing ingredient is an upper bound on the maximum average KL divergence between $\mathbb{P}_{\theta_i}$ and $\mathbb{P}_0$: we can obtain it using Lemma 6.B.6. We only need to assume

$$\|\theta_i\|_F \leq r\sqrt{Ds} \leq \min\left\{\vartheta, \frac{(1-\vartheta)^2(\sigma_{\min}^2 + \omega^2)}{4\sigma_{\max}^2}\right\}$$

to get the upper bound

$$\begin{aligned}\max_i \mathbb{E}_\Pi\left[\mathrm{KL}\left\{\mathbb{P}_{\theta_i}(Y \mid \Pi) \parallel \mathbb{P}_{\theta_0}(Y \mid \Pi)\right\}\right] &\leq \max_i \mathrm{KL}_{\mathrm{avg}}(\|\theta_i\|_2, \|\theta_i\|_F) \\ &\leq \mathrm{KL}_{\mathrm{avg}}(rs, r\sqrt{Ds}).\end{aligned}$$

Since we must satisfy the constraint from Equation (A.1) in Fano's method, we will choose r so that:

$$\mathrm{KL}_{\mathrm{avg}}(rs, r\sqrt{Ds}) \leq \alpha \log(M) = \alpha \log\left(2^{Ds/8} - 1\right)$$

with $\alpha = \frac{\log 3 - \log 2}{2 \log 2}$. We want to solve the previous inequality for r, and for that we start by replacing $\mathrm{KL}_{\mathrm{avg}}(rs, r\sqrt{Ds})$ with its value from Lemma 6.B.6, replacing γ_ℓ with γ_ℓ to lighten notations:

$$\begin{aligned}\mathrm{KL}_{\mathrm{avg}}(rs, r\sqrt{Ds}) \leq \alpha \log\left(2^{Ds/8} - 1\right) &\iff \frac{2}{\gamma_\ell} Tp\left((rs)^2 + q_\ell\right)(r\sqrt{Ds})^2 \leq cDs \\ &\iff Ds^3 r^4 + q_\ell Dsr^2 - c\frac{\gamma_\ell^2 Ds}{Tq_\ell} \leq 0.\end{aligned}$$

If we consider this as a degree two polynomial in the variable r^2, its determinant is

$$\Delta = q_\ell^2 D^2 s^2 + 4Ds^3 c\frac{\gamma_\ell^2 Ds}{Tp}.$$

For β to be small enough, r^2 must remain below the only positive root of the polynomial, namely

$$r^2 \leq \frac{-q_\ell Ds + \sqrt{q_\ell^2 D^2 s^2 + c\frac{\gamma_\ell^2 D^2 s^4}{Tp}}}{2Ds^3} = \frac{q_\ell}{2s^2}\left(\sqrt{1 + c\frac{\gamma_\ell^2 s^2}{Tpq_\ell^2}} - 1\right).$$

If we assume the quantity $c\frac{\gamma_\ell^2 s^2}{Tpq_\ell^2}$ inside the square root is smaller than 1, i.e.

$$\frac{\gamma_\ell s}{\sqrt{p} q_\ell \sqrt{T}} \leq c, \tag{6.25}$$

then we can lower-bound $\sqrt{1+x}$ by its chord $(\sqrt{2}-1)x$. In other words, a sufficient condition for r^2 to remain small enough is given by

$$r^2 \leq \frac{q_\ell}{2s^2} \times (\sqrt{2}-1)c\frac{\gamma_\ell^2 s^2}{Tpq_\ell^2} = c\frac{\gamma_\ell^2}{Tpq_\ell}.$$

To sum up, we have three constraints on r:

$$rs \leq \vartheta \qquad\qquad rs \leq \frac{(1-\vartheta)^2(\sigma_{\min}^2 + \omega^2)}{4\sigma_{\max}^2} = \frac{\sqrt{1-\vartheta}}{4}\gamma_\ell \qquad\qquad r \leq \sqrt{c\frac{\gamma_\ell^2}{Tpq_\ell}}.$$

We can therefore choose r as the largest value satisfying all three of them:

$$r = \frac{1}{s}\min\left\{\vartheta, \frac{\gamma_\ell\sqrt{1-\vartheta}}{4}, c\frac{\gamma_\ell s}{\sqrt{Tpq_\ell}}\right\} \tag{6.26}$$

To reach our conclusion, we simply need to remark that the vectorized ℓ_1 distance between any two matrices in $\mathcal{K}(r)$ gives us a lower bound on the operator ℓ_∞ distance that separates them:

$$\begin{aligned}\|\theta_i - \theta_j\|_\infty &= \max_{k\in[D]} \sum_{l\in[D]} |(\theta_i - \theta_j)|_{k,l} \geq \frac{1}{D}\sum_{1\leq k,l\leq D} |(\theta_i - \theta_j)|_{k,l} \\ &= \frac{1}{D}\|\mathrm{vec}(\theta_i) - \mathrm{vec}(\theta_j)\|_1 \geq \frac{rDs}{8D} = \frac{rs}{8}\end{aligned}$$

Subsequently, our parameters θ_i are 2τ-separated (in ℓ_∞ operator distance) with $\tau = rs/8$. As soon as the minimum in Equation (6.26) is reached by the third value, i.e. whenever

$$\frac{\gamma_\ell s}{\sqrt{Tpq_\ell}} \leq c\min\{\vartheta, \gamma_\ell\sqrt{1-\vartheta}\} \tag{6.27}$$

we can simplify the expression of τ:

$$\tau = c\frac{\gamma_\ell s}{\sqrt{Tpq_\ell}}.$$

In this case, by Lemma A.2.1, we can conclude:

$$\inf_{\widehat{\theta}} \sup_{\theta\in\Theta_s} \mathbb{P}_\theta\left[\|\widehat{\theta} - \theta\|_\infty \geq c\frac{\gamma_\ell s}{\sqrt{Tpq_\ell}}\right] \geq \frac{\log(M+1) - \log 2}{\log M} - \alpha \geq \frac{1}{2}.$$

□

7

Learning with combinatorial optimization layers: a probabilistic approach

You are now creating six different timelines.

Abed Nadir
Community – S3E4
Remedial Chaos Theory (2011)

Contents

This chapter corresponds to our preprint D., Baty, et al. (2022) and was presented at JuliaCon 2022 (D., Bouvier, and Baty 2022)

7.1 Introduction

ML and CO are two essential ingredients of modern industrial processes. While ML extracts meaningful information from noisy data, CO enables decision-making in high-dimensional constrained environments. But in many situations, combining both of these tools is necessary: for instance, we might want to generate predictions from data, and then use those predictions to make optimized decisions. To do that, we need *pipelines* that contain two types of *layers*: ML layers and CO layers.

Due to their many possible applications, hybrid ML-CO pipelines currently attract a lot of research interest. The recent reviews by Bengio, Lodi, and Prouvost (2021) and Kotary et al. (2021) are excellent resources on this topic. Unfortunately, relevant software implementations are scattered across paper-specific repositories, with few tests, minimal documentation and sporadic code maintenance. Not only does this make comparison and evaluation difficult for academic purposes, it also hurts practitioners wishing to experiment with such techniques on real use cases.

Let us discuss a generic hybrid ML-CO pipeline, which includes a CO oracle amid several ML layers:

$$\xrightarrow{\text{Input } x} \boxed{\text{ML layers}} \xrightarrow{\text{Objective } \theta} \boxed{\boxed{\text{CO oracle}}} \xrightarrow{\text{Solution } y} \boxed{\text{More ML layers}} \xrightarrow{\text{Output}} \tag{7.1}$$

The inference problem consists in predicting an output from a given input. It is solved online, and requires the knowledge of the parameters for each ML layer. On the other hand, the learning problem aims at finding parameters that lead to "good" outputs during inference. It is solved offline based on a training set that contains several inputs, possibly complemented by target outputs.

In Equation (7.1), we use the term *CO oracle* to emphasize that any algorithm may be used to solve the optimization problem, whether it relies on an existing solver or a handcrafted implementation. Conversely, when we talk about a *layer*, it is implied that we can compute meaningful derivatives using Automatic Differentiation (AD). Since it may call black box subroutines, an arbitrary CO oracle is seldom compatible with AD. And even when it is, its derivatives are zero almost everywhere, which gives us no exploitable slope information. Therefore, according to our terminology, *a CO oracle is not a layer (yet)*, and the whole point of this paper is to turn it into one.

Modern ML libraries provide a wealth of basic building blocks that allow users to assemble and train complex pipelines. We want to leverage these libraries to create hybrid ML-CO pipelines, but we face two main challenges. First, while ML layers are easy to construct, it is not obvious how to transform a CO oracle into a usable layer. Second, standard ML losses are ill-suited to our setting, because they often ignore the underlying optimization problem.

Our goal is to remove these difficulties. We introduce `InferOpt.jl`[1], a Julia package which 1) can turn any CO oracle into a layer with meaningful derivatives, and 2) provides structured loss functions that work well with the resulting layers. It contains several state-of-the-art methods that

[1] https://github.com/axelparmentier/InferOpt.jl

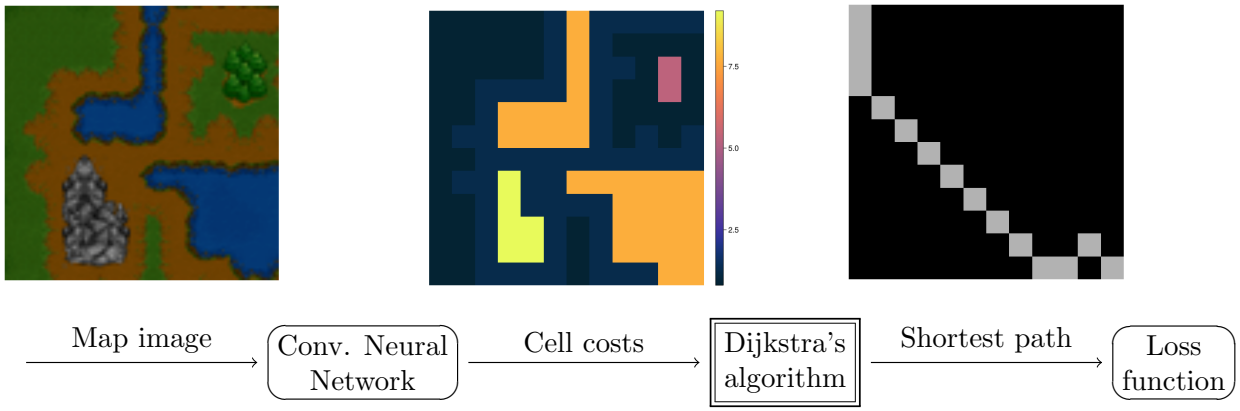


Figure 7.1: Pipeline for computing shortest paths on Warcraft maps
Data from Vlastelica et al. (2020)

are fully compatible with Julia's AD and ML ecosystem, making CO layers as easy to use as any ML layer. To describe the available methods in a coherent manner, we leverage the unifying concept of probabilistic CO layer, hence the name of the package.

7.1.1 Motivating example

Let us start by giving an example of hybrid ML-CO pipeline. Suppose we want to find shortest paths on a map, but we do not have access to an exact description of the underlying terrain. Instead, all we have are images of the area, which give us a rough idea of the topography and obstacles. To solve our problem, we need a pipeline comprising two layers of very different natures. First, an image processing layer, which is typically implemented as a Convolutional Neural Network (CNN). The CNN is tasked with translating the images into a weighted graph. Second, a CO layer performing shortest path computations on said weighted graph (*e.g.*, using Dijkstra's algorithm).

This pipeline is exactly the one considered by Vlastelica et al. (2020) and Berthet et al. (2020) for pathfinding on video game maps. We illustrate it on Figure 7.1, and we describe it in more detail in Section 7.2.3. The goal is to learn appropriate weights for the CNN, so that it feeds accurate cell costs to Dijkstra's algorithm. This is done by minimizing a loss function, such as the distance between the true optimal path and the one we predict.

7.1.2 Our setting

In our hybrid ML-CO pipelines, we consider CO oracles f that solve the following kind of problem:

$$f : \theta \longmapsto \underset{v \in \mathcal{V}}{\operatorname{argmax}}\, \theta^\top v \tag{7.2}$$

Here, the input $\theta \in \mathbb{R}^d$ is the *objective direction*. Meanwhile, $\mathcal{V} \subset \mathbb{R}^d$ (for *vertices*) denotes a finite set of feasible solutions – which may be exponentially large in d – among which the optimal solution $f(\theta)$ shall be selected. For simplicity, we assume that f is single-valued, *i.e.*, that the optimal solution is unique.

The feasible set $\mathcal{V}$ and its dimension may depend on the instance. For instance, if Equation (7.2) is a shortest path problem, the underlying graph may change from one input to another. If we

wanted to remain generic, we should therefore write $\mathcal{V}(x) \subset \mathbb{R}^{d(x)}$. To keep notations simple, we omit the dependency in x whenever it is clear from the context. Note that we could also study more general CO oracles given by

$$\underset{v \in \mathcal{V}}{\operatorname{argmax}}\, \theta^\top g(v)$$

where g is any function from an arbitrary finite set $\mathcal{V}$ to $\mathbb{R}^d$. As long as the objective is linear in θ, the theory we present generalizes seamlessly. However, for ease of exposition, we keep $g(v) = v$. In this case, Equation (7.2) is equivalent to

$$\underset{v \in \operatorname{conv}(\mathcal{V})}{\operatorname{argmax}}\, \theta^\top v.$$

Indeed, when the objective is linear in v, it makes no difference to optimize over the convex hull $\operatorname{conv}(\mathcal{V})$ instead of optimizing on $\mathcal{V}$.

7.1.2.1 From an optimization problem to an oracle

It is important to note that the formulation $\operatorname{argmax}_{v \in \mathcal{V}} \theta^\top v$ is very generic. Any Linear Program (LP) or Integer Linear Program (ILP) can be written this way, as long as its feasible set is bounded. Indeed, the optimum of an LP is always reached at a vertex of the polytope, of which there are finitely many. Similarly, the optimum of an ILP is always reached at an integral point, or even better, at a vertex of the convex hull of the integral points. As a result, Equation (7.2) encompasses a variety of well-known CO problems related to graphs (paths, flows, spanning trees, coloring), resource management (knapsack, bin packing), scheduling, *etc.* See Korte and Vygen (2006) for an overview of CO and its applications.

For every one of these problems, dedicated algorithms have been developed over the years, which sometimes exploit the domain structure better than a generic ILP library (such as Gurobi or SCIP). Thus, we have no interest in restricting the procedure used to compute an optimal solution: we want to pick the best solver for each application. That is why the methods discussed in this paper only need to access the CO oracle f as a black box function, without making assumptions on its implementation.

7.1.2.2 From an oracle to a probability distribution

When using CO oracles within ML pipelines, the first challenge we face is the *lack of useful derivatives*. Since training often relies on Stochastic Gradient Descent (SGD), we need to be able to backpropagate loss gradients onto the weights of the ML layers. This requires each individual layer to be differentiable, so that we can compute the Jacobian of its output with respect to its input. Unfortunately, since the feasible set $\mathcal{V}$ of Equation (7.2) is finite, the CO oracle is a piecewise constant mapping and its derivatives are zero almost everywhere. To recover useful slope information, we seek *approximate derivatives*, which is where the probabilistic approach comes into play.

To describe it, we no longer think about a CO oracle as a function returning a single element $f(\theta)$ from $\mathcal{V}$. Instead, we use it to define a probability distribution $p(\cdot|\theta)$ on $\mathcal{V}$. The naive choice would be the Dirac mass $p(v|\theta) = \delta_{f(\theta)}(v)$, but it shares the lack of differentiability of the oracle itself. Thus, our goal is to spread out the distribution p into an approximation $\widehat{p}$, such that the probability mapping $\theta \longmapsto \widehat{p}(\cdot|\theta)$ becomes smooth with respect to θ. If we can do that, then the expectation

mapping

$$\widehat{f}\colon \theta \longmapsto \mathbb{E}_{\widehat{p}(\cdot|\theta)}[V] = \sum_{v \in \mathcal{V}} v\widehat{p}(v|\theta), \tag{7.3}$$

where it is understood that $V \sim \widehat{p}(\cdot|\theta)$, will be just as smooth. This expectation mapping $\widehat{f}$ is what we take to be our *probabilistic CO layer*: see Section 7.3 for detailed examples. In what follows, plain letters (p, f) always refer to the initial CO oracle, while letters with a hat $(\widehat{p}, \widehat{f})$ refer to the probabilistic CO layer that we wrap around it.

7.1.2.3 From a probability distribution to a loss function

The presence of CO oracles in ML pipelines gives rise to a second challenge: the choice of an appropriate loss function to learn the parameters. As highlighted by Bengio, Lodi, and Prouvost (2021), this choice heavily depends on the data at our disposal. They distinguish two main paradigms, which we illustrate using the pipeline of Figure 7.1.

If our dataset only contains the map images, then we are in a weakly supervised setting, which they call *learning by experience* (see Section 7.4). In that case, the loss function will evaluate the solutions computed by our pipeline using the true cell costs. On the other hand, if our dataset happens to contain precomputed targets such as the true shortest paths, then we are in a fully supervised setting, which they call *learning by imitation* (see Section 7.5). In that case, the loss function will compare the paths computed by our pipeline with the optimal ones, hoping to minimize the discrepancy. For both of these cases, the probabilistic perspective plays an important role in ensuring smoothness of the loss.

7.1.2.4 Complete pipeline

The typical pipeline we will focus on is a special case of Equation (7.1), which we now describe in more detail:

$$\xrightarrow{\text{Input } x} \boxed{\text{ML layer } \varphi_w} \xrightarrow{\text{Objective } \theta = \varphi_w(x)} \boxed{\boxed{\text{CO oracle } f}} \xrightarrow{\text{Solution } y = f(\theta)} \boxed{\text{Loss function } \mathcal{L}} \tag{7.4}$$

Our pipeline starts with an ML layer φ_w, where w stands for the vector of parameters (or weights). Its role is to encode the input x into an objective direction $\theta = \varphi_w(x)$, which is why we often refer to it as the *encoder*. Then, the CO oracle defined in Equation (7.2) returns an optimal solution $y = f(\theta)$. Finally, the loss function $\mathcal{L}$ is used during training to evaluate the quality of the solution. If our dataset contains N input samples $x^{(1)}, ..., x^{(N)}$, learning involves applying SGD to the following loss minimization problem:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\Big(\overbrace{f\big(\underbrace{\varphi_w(x^{(i)})}_{\theta^{(i)}}\big)}^{y^{(i)}}, \dots \Big). \tag{7.5}$$

The dots ... correspond to additional arguments that may be used by the loss. For instance, the loss may depend on the input $x^{(i)}$ itself, or require target outputs $\bar{t}^{(i)}$ for comparison (see Section 7.5).

Remark 7.1.1. *Our work focuses on individual layers and loss functions. Although we present various concrete examples, we do not give generic advice on how to build the whole pipeline for a specific application. If the use case corresponds to a "Predict, then Optimize" setting such as the one from Figure 7.1, then Elmachtoub and Grigas (2022) give a few useful pointers. If the goal is to*

approximate hard optimization problems with easier ones, the reader can refer to Parmentier (2021a) for a general methodology.

7.1.3 Contributions

Our foremost contribution is the open-source package `InferOpt.jl`, which is written in the Julia programming language (Bezanson et al. 2017). Given a CO oracle provided as a callable object, our package wraps it into a probabilistic CO layer that is compatible with Julia's AD and ML ecosystem. This is achieved thanks to the `ChainRules.jl`[2] interface (White et al. 2022): see Chapter 3 for details. Moreover, `InferOpt.jl` defines several structured loss functions, both for learning by experience and for learning by imitation.

On top of that, we present theoretical insights that fill some gaps in previous works. In addition to the framework of probabilistic CO layers, we propose:

- A new perturbation technique designed for CO oracles that only accept objective vectors with a certain sign (such as Dijkstra's algorithm, which fails on graphs with negative edge costs): see Section 7.3.3.2.
- A way to differentiate through a large subclass of probabilistic CO layers (those that rely on convex regularization) by combining the Frank-Wolfe algorithm with implicit differentiation: see Section 7.3.3.3.
- A probabilistic regularization of the regret for learning by experience: see Section 7.4.
- A generic decomposition framework for imitation losses (similar to the cost-sensitive Fenchel-Young losses of Blondel, Martins, and Niculae (2020)), which subsumes most of the literature so far and suggests ways to build new loss functions: see Section 7.5.1.

Finally, we describe numerical experiments on our motivating example of Warcraft shortest paths. In particular, we use the pipeline of Figure 7.1 for learning by experience, even though the CNN encoder has tens of thousands of parameters. To the best of our knowledge, previous attempts to learn hybrid ML-CO pipelines by experience were restricted to ML layers with fewer than 100 parameters.

7.1.4 Outline

In Section 7.2, we review the literature on differentiable optimization layers, before focusing on the Warcraft example. Section 7.3 introduces the family of probabilistic CO layers by splitting it into perturbed and regularized approaches. Then, Section 7.4 gives tools for learning by experience, while Section 7.5 discusses loss functions for learning by imitation. An application of our package to shortest paths on video game maps is presented in Section 7.6.

[2] https://github.com/JuliaDiff/ChainRules.jl

7.2 Related work

7.2.1 Optimization layers in ML

A significant part of modern ML relies on AD: see Baydin et al. (2018) for an overview and Griewank and Walther (2008) for an in-depth treatment. In particular, AD forms the basis of the backpropagation algorithm used to train neural networks.

7.2.1.1 The notion of implicit layer

Standard neural architectures draw from a small collection of *explicit* layers (Goodfellow, Bengio, and Courville 2016). Whatever their connection structure (dense, convolutional, recurrent, *etc.*) and regardless of their activation function, these layers all correspond to input-output mappings that can be expressed using an analytic formula. This same formula is then used by AD to compute gradients.

On the other hand, the layers defined by `InferOpt.jl` are of the *implicit* kind, which means they can contain arbitrarily complex iterative procedures. While we focus here on optimization algorithms, those are not the only kind of implicit layers: fixed point iterations and differential equation solvers are also widely used, depending on the application at hand. See the tutorial by Kolter, Duvenaud, and M. Johnson (2020) for more thorough explanations.

Due to the high memory cost of unrolling iterative procedures, efficient AD of implicit layers often relies on the implicit function theorem. As long as we can specify a set of conditions satisfied by the input-output pair, this theorem equates differentiation with solving a linear system of equations. See the Python package `jaxopt`[3] for an example implementation, and its companion paper for theoretical details (Blondel, Berthet, et al. 2022). The recent Python package `theseus`[4] showcases an application of this framework to robotics and vision (Pineda et al. 2022).

7.2.1.2 Convex optimization layers

Among the early works on optimization layers for deep learning, the seminal OptNet paper by Amos and Kolter (2017) stands out. It describes a way to differentiate through quadratic programs (QPs) by using the Karush-Kuhn-Tucker (KKT) optimality conditions and plugging them into the implicit function theorem.

More sophisticated tools exist for disciplined conic programs, such as the Python package `cvxpylayers`[5] (Agrawal, Amos, et al. 2019). The recent Julia package `DiffOpt.jl`[6] (Sharma et al. 2022) extends these ideas beyond the conic case to general convex programs. Note that both libraries only accept optimization problems formulated in a domain-specific modeling language, as opposed to arbitrary oracles.

Strong convexity makes differentiation easier because the solutions evolve smoothly as a function of the constraints and objective parameters. In particular, this means the methods listed above return exact derivatives and do not rely on approximations. Regrettably, this nice behavior falls apart as soon as we enter the combinatorial world.

[3] https://github.com/google/jaxopt
[4] https://github.com/facebookresearch/theseus
[5] https://github.com/cvxgrp/cvxpylayers
[6] https://github.com/jump-dev/DiffOpt.jl

7.2.1.3 Linear optimization layers

Let us consider an LP whose feasible set is a bounded polyhedron, also called *polytope*. It is well-known that for generic objective directions, the optimal solution will be unique and located at a vertex of the polytope. Even though LPs look like continuous optimization problems, this property shows that they are fundamentally combinatorial. Indeed, a small change in the objective direction can cause the optimal solution to suddenly jump to another vertex, which results in a discontinuous mapping from objectives to solutions. In fact, this mapping is piecewise constant, which means no useful differential information can come from it: its Jacobian is undefined at the jump points and zero everywhere else.

Therefore, when differentiating LPs with respect to their objective parameters, we need to resort to approximations. Vlastelica et al. (2020) use interpolation to turn a piecewise constant mapping into a piecewise linear and continuous one. However, the dominant approximation paradigm in the literature is regularization, as formalized by Blondel, Martins, and Niculae (2020).

For instance, Wilder, Dilkina, and Tambe (2019) add a quadratic penalty to the linear objective, which allows them to reuse the QP computations of Amos and Kolter (2017). Mandi and Guns (2020) propose a log-barrier penalty, which lets them draw a connection with interior-point methods. Berthet et al. (2020) suggest perturbing the optimization problem by adding stochastic noise to the objective direction, which is a form of implicit regularization.

When LP layers are located at the end of a pipeline, a clever choice of loss function can also simplify differentiation. This is illustrated by the Structured Support Vector Machine (S-SVM) loss (Tsochantaridis et al. 2005), Smart "Predict, then Optimize" (SPO+) loss (Elmachtoub and Grigas 2022) and Fenchel-Young (FY) loss (Blondel, Martins, and Niculae 2020).

7.2.1.4 Integer optimization layers

In theory, the methods from the previous section still work in the presence of integer variables, that is, for ILPs. To apply them, we only need to consider the polytope defined by the convex hull of integral solutions. Alas, in the general case, there is no concise way to describe this convex hull. This is why many authors decide to differentiate through the continuous relaxation of the ILP instead (Mandi, Demirović, et al. 2020): it is an outer approximation of the integral polytope, but it can be sufficient for learning purposes. Some suggest taking advantage of techniques specific to integer programming, such as integrality cuts (Ferber et al. 2020) or a generalization of the notion of active constraint (Paulus et al. 2021).

There has also been significant progress on finding gradient approximations for combinatorial problems such as ranking (Blondel, Teboul, et al. 2020) and shortest paths (Parmentier 2021b). Yet these techniques are problem-specific, and therefore hard to generalize, which is why we leave them aside.

Instead, we want to allow implicit manipulation of the integral polytope itself, without making assumptions on its structure. To achieve that, we can only afford to invoke CO oracles as black boxes (Vlastelica et al. 2020; Berthet et al. 2020). This compatibility with arbitrary algorithms is one of the fundamental tenets of our work. The recent Python package `PyEPO`[7] (Tang and Khalil 2022) shares this perspective, but it only implements a subset of `InferOpt.jl` since it does not address the central notion of probabilistic CO layer.

[7] https://github.com/khalil-research/PyEPO

7.2.2 Similarities and differences with reinforcement learning

As we will see in Section 7.4, learning by experience can be reminiscent of Reinforcement Learning (RL), which also relies on a reward or cost signal given by the environment (Sutton and Barto 2018). Furthermore, the encoder layer of Equation (7.4) is similar to a parametric approximation of the value function, which forms the basis of deep RL approaches. This prompts us to discuss a few differences between RL and the framework we study.

The standard mathematical formulation of RL is based on Markov decision processes, where a reward and state transition are associated with each action. Usually, the available actions are elementary decisions: pull one lever of a multi-armed bandit, cross one edge on a graph, select one move in a board game, *etc.* The resulting value or policy update is local: it is specific to both the current state and the action taken. The more reward information we gather, the more efficient learning becomes.

In our framework, the basic step is a call to the optimizer. But combinatorial algorithms can go beyond simple actions: they often output a structured and high-dimensional solution, which aggregates many elementary decisions. This in turn triggers a global update in our knowledge of the system, whereby the final reward is redistributed between all elementary decisions. In a way, *backpropagation through the optimizer enables efficient credit assignment*, even for sparse reward signals.

Another difference is related to the Bellman fixed point equation. In an RL setting, the Bellman equation is used explicitly to derive parameter updates. In our setting, the Bellman equation is used implicitly within optimizers such as Dijkstra's algorithm.

To conclude, while standard RL decomposes a policy into elementary decisions, the pipelines we study here are able to look directly for complex multistep solutions. Note that a similar concept of *option* exists in hierarchical RL (Barto and Mahadevan 2003): comparing both perspectives in detail would no doubt be fruitful, and we leave it for future work.

7.2.3 Our guiding example: shortest paths on Warcraft maps

As a way to clarify the concepts introduced in this paper, we illustrate them on the problem of Warcraft shortest paths (Vlastelica et al. 2020; Berthet et al. 2020), which was already introduced on Figure 7.1. The associated dataset[8], assembled by Vlastelica et al. (2020), contains randomly-generated maps similar to those from the Warcraft II video game. Each of these maps is a red-green-blue (RGB) image of size $ks \times ks$ containing $k \times k$ square cells of side length s. Every cell has its own terrain type (grass, forest, water, earth, *etc.*) which incurs a specific cost when game characters cross it.

The goal is to find the shortest path from the top left corner of the map to the bottom right corner. At prediction time, cell costs are unknown, which means they must be approximated from the image alone. Solving the problem of Warcraft shortest paths thus requires adapting the pipeline of Equation (7.4) as follows:

$$\xrightarrow[x \in \mathbb{R}^{ks \times ks \times 3}]{\text{Map image}} \boxed{\text{Conv. Neural Network } \varphi_w} \xrightarrow[\theta \in \mathbb{R}^{k \times k}]{\text{Negative cell costs}} \boxed{\boxed{\text{Dijkstra's algorithm } \operatorname{argmax}_{v \in \mathcal{P}_k} \theta^\top v}} \xrightarrow[y \in \mathcal{P}_k]{\text{Shortest path}} \boxed{\text{Loss function } \mathcal{L}} \tag{7.6}$$

[8] https://edmond.mpdl.mpg.de/dataset.xhtml?persistentId=doi:10.17617/3.YJCQ5S

```julia
using Flux, Metalhead, Statistics

resnet18 = ResNet(
    18; pretrain=false, nclasses=1
)

warcraft_encoder = Chain(
    resnet18.layers[1][1:4],
    AdaptiveMaxPool((12, 12)),
    x -> mean(x; dims=3),
    x -> dropdims(x; dims=(3, 4)),
    x -> -softplus.(x)
)
```

Code sample 7.1: CNN encoder for Warcraft

```julia
using Graphs, GridGraphs, LinearAlgebra

function warcraft_maximizer(theta)
    g = GridGraph(-theta)
    path = grid_dijkstra(g, 1, nv(g))
    y = path_to_matrix(g, path)
    return y
end

function warcraft_cost(y; theta_ref)
    return dot(y, theta_ref)
end
```

Code sample 7.2: Dijkstra optimizer for Warcraft

where we have defined the set of feasible paths

$$\mathcal{P}_k = \{v \in \{0,1\}^{k \times k} : \ v \text{ represents a path from } (1,1) \text{ to } (k,k)\}.$$

Training proceeds based on the map images and (possibly) the true shortest paths or cell costs provided in the dataset.

As suggested by Vlastelica et al. (2020), we design the CNN based on the first few layers of a ResNet18 (He et al. 2016). We also append a negative softplus activation, in order to make sure that all outputs are negative. The sign constraint is there to ensure that Dijkstra's algorithm will terminate, but it is not obvious why we require negative costs instead of positive ones. The reason behind this sign switch is that Dijkstra's algorithm is a minimization oracle, whereas by convention `InferOpt.jl` works with maximization oracles.

We now show how to implement this pipeline in Julia. Throughout the paper, in addition to `InferOpt.jl`, the following packages are used: `Flux.jl`[9] (Innes et al. 2018; Innes 2018), `Graphs.jl`[10] (Fairbanks et al. 2021), `GridGraphs.jl`[11], `Metalhead.jl`[12] and `Zygote.jl`[13] (Innes 2019), along with the standard libraries `LinearAlgebra` and `Statistics`. Code sample 7.1 creates the CNN encoder layer. Meanwhile, Code sample 7.2 shows how to define the Dijkstra oracle and the true cost function.

Finally, Code sample 7.3 demonstrates the full prediction and optimization pipeline. It assumes that we have already parsed the Warcraft dataset into three lists:

- `images`, whose elements are three-dimensional arrays representing map images;
- `cells`, whose elements are matrices representing true cell costs;
- `paths`, whose elements are binary matrices representing true shortest paths.

[9] https://github.com/FluxML/Flux.jl
[10] https://github.com/JuliaGraphs/Graphs.jl
[11] https://github.com/gdalle/GridGraphs.jl
[12] https://github.com/FluxML/Metalhead.jl
[13] https://github.com/FluxML/Zygote.jl

```julia
x, theta_ref, y_ref = images[1], cells[1], paths[1]
theta = warcraft_encoder(x)
y = warcraft_maximizer(theta)
c = warcraft_cost(y; theta_ref=theta_ref)
```

Code sample 7.3: Full pipeline for Warcraft shortest paths

These functions do not rely on `InferOpt.jl`, but they will be used throughout the paper inside the differentiable wrappers provided by the package. Note that some snippets shown here have been shortened for clarity. Please refer to the documentation of `InferOpt.jl` and satellite packages for actual examples you can run.

7.3 Probabilistic CO layers

In this section, we focus on the CO oracle f defined in Equation (7.2), which is piecewise constant. By adopting a probabilistic point of view, we construct several smooth approximations $\widehat{f}$, which can be computed and differentiated based solely on calls to f.

7.3.1 The expectation of a differentiable probability distribution

As announced in Section 7.1.2.2, a probabilistic CO layer works in two steps. First, it constructs a probability distribution $\widehat{p}(\cdot|\theta) \in \Delta^{\mathcal{V}}$. Second, it returns the expectation $\widehat{f}(\theta) = \mathbb{E}_{\widehat{p}(\cdot|\theta)}[V] \in \mathrm{conv}(\mathcal{V})$. Figure 7.2 illustrates this behavior on a two-dimensional polytope, with a maximization problem defined by the vector θ (black arrow). While the CO oracle outputs a single optimal vertex (red square), the probabilistic CO layer defines a distribution on all the vertices (light blue circles). Its output (dark blue hexagon) is a convex combination of the vertices with nonzero weights, which belongs to the convex hull of $\mathcal{V}$ (gray surface).

For such a layer to be useful in our setting, we impose several conditions. First, all computations must only require calls to the CO oracle f. Second, the expectation $\mathbb{E}_{\widehat{p}(\cdot|\theta)}[V]$ must be tractable (whether it is with an explicit formula, Monte-Carlo sampling, variational inference, *etc.*). Third, the mapping $\theta \longmapsto \widehat{p}(\cdot|\theta)$ must be differentiable. If these conditions are satisfied, then the Jacobian of $\widehat{f}$ is easily deduced from Equation (7.3):

$$\mathrm{J}_\theta \widehat{f}(\theta) = \mathrm{J}_\theta \mathbb{E}_{\widehat{p}(\cdot|\theta)}[V] = \sum_{v \in \mathcal{V}} v \nabla_\theta \widehat{p}(v|\theta)^\top \tag{7.7}$$

Let us give an example where analytic formulas exist. If $\mathcal{V} = \{\mathrm{e}_1, ..., \mathrm{e}_d\}$ is the set of basis vectors, then its convex hull $\mathrm{conv}(\mathcal{V}) = \Delta^d$ is the unit simplex of dimension d. Given an objective direction θ, solving $\mathrm{argmax}_{v \in \mathcal{V}} \theta^\top v$ yields the basis vector $f(\theta) = \mathrm{e}_i$ where i is the index maximizing θ_i.

We want a probability distribution that evolves smoothly with θ, so we need to spread out the naive Dirac mass $\delta_{f(\theta)}(\cdot)$ by putting weight on several vertices instead of just one. Let us assign to each vertex a probability that depends on its level of optimality in the optimization problem $\mathrm{argmax}_{v \in \mathcal{V}} \theta^\top v$, that is, on its inner product with θ. The Boltzmann distribution is a natural candidate, leading to $\widehat{p}(\mathrm{e}_i|\theta) \propto e^{\theta^\top \mathrm{e}_i} = e^{\theta_i}$. Computing the expectation reveals a well-known

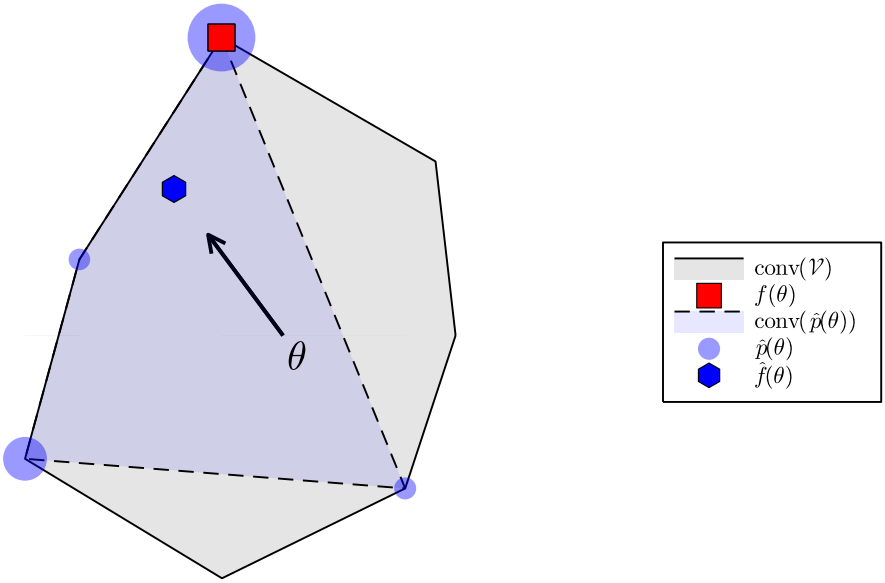


Figure 7.2: Effect of a probabilistic CO layer

operation:

$$\widehat{f}(\theta) = \mathbb{E}_{\widehat{p}(\cdot|\theta)}[V] = \sum_{i=1}^{d} \frac{e^{\theta_i}}{\sum_{j=1}^{d} e^{\theta_j}} \mathrm{e}_i = \mathrm{softmax}(\theta)$$

Unlike the "hardmax" function f, the softmax function $\widehat{f}$ is differentiable, which justifies its frequent use as an activation function in classification tasks.

While the Boltzmann distribution can be used for specific sets $\mathcal{V}$, in the general case, it has an intractable normalizing constant. This means we would need MCMC methods to compute derivatives, which undermines the simplicity we are looking for. Fortunately, Sections 7.3.3.1 and 7.3.3.2 present other probability distributions which can be easily approximated through sampling.

7.3.2 Regularization as another way to define a distribution

Although this probabilistic point of view was recently put forward by Berthet et al. (2020), the most popular paradigm in the literature remains regularization (Blondel, Martins, and Niculae 2020). Instead of using the CO oracle (7.2), regularization solves a different problem:

$$\widehat{f}_\Omega : \theta \longmapsto \operatorname*{argmax}_{\mu \in \mathrm{dom}(\Omega)} \theta^\top \mu - \Omega(\mu) \tag{7.8}$$

where $\Omega : \mathbb{R}^d \to \mathbb{R}$ is a smooth and convex function that penalizes the output μ. Usually, Ω is chosen to enforce $\Omega(\mu) = +\infty$ whenever $\mu \notin \mathrm{conv}(\mathcal{V})$, which means $\mathrm{dom}(\Omega) \subseteq \mathrm{conv}(\mathcal{V})$. Remember that since $\mathcal{V}$ is finite, $\mathrm{conv}(\mathcal{V})$ is a polytope whose vertices form a subset of $\mathcal{V}$.

The change of notation from v to μ stresses the fact that v is an element of $\mathcal{V}$, while μ belongs to $\mathrm{dom}(\Omega) \subseteq \mathrm{conv}(\mathcal{V})$. By definition of the convex hull, any feasible μ is the expectation of some

Layer	Notations	Probability $\widehat{p}(\cdot\|\theta)$	Regularization
`PerturbedAdditive`	$\widehat{p}_\varepsilon^+$, $\widehat{f}_\varepsilon^+$	Explicit: $f(\theta + \varepsilon Z)$	Implicit: Fenchel conjugate
`PerturbedMultiplicative`	$\widehat{p}_\varepsilon^\odot$, $\widehat{f}_\varepsilon^\odot$	Explicit: $f(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2})$	Implicit: Fenchel conjugate
`RegularizedGeneric`	$\widehat{p}_\Omega^{\text{FW}}$, $\widehat{f}_\Omega^{\text{FW}}$	Implicit: Frank-Wolfe weights	Explicit: function Ω

Table 7.1: Probabilistic CO layers and their defining features

distribution over $\mathcal{V}$, hence our choice of letter. This means we can write $\widehat{f}_\Omega(\theta)$ as a convex combination of the elements of $\mathcal{V}$, whose weights are then interpreted as probabilities $\widehat{p}_\Omega(\cdot|\theta)$. In other words, the two perspectives are not opposed: regularization is just another way to define a probability distribution.

For instance, if we go back to the concrete example from Section 7.3.1 and select $\Omega(\mu) = \sum_i \mu_i \log \mu_i$ (the negative Shannon entropy), we find once again that $\widehat{f}_\Omega(\theta) = \text{softmax}(\theta)$. But the case of the unit simplex is very peculiar, because for any $\mu \in \text{conv}(\mathcal{V})$, the convex decomposition of μ onto the vertices $\mathcal{V}$ is unique. In other words, the correspondence between a regularization Ω and a probability mapping $\widehat{p}_\Omega$ is one-to-one.

This does not hold for arbitrary polytopes. As a result, we need to be more specific in how we choose the convex decomposition. In particular, we need the weights to be differentiable, in order to compute the Jacobian with Equation (7.7). Section 7.3.3.3 describes one possible approach, which relies on the Frank-Wolfe algorithm and implicit differentiation.

Conversely, the probability distributions given in Sections 7.3.3.1 and 7.3.3.2 also give rise to an implicit regularization, which can be expressed using Fenchel conjugates. This point of view is especially useful when we want to combine these layers with Fenchel-Young losses (Section 7.5.2.3). In essence, we claim that *probabilistic CO layers and regularization are two sides of the same coin.*

7.3.3 Collection of probabilistic CO layers

Our package implements various flavors of probabilistic CO layers, which are summed up in Table 7.1. Code sample 7.4 displays the operations they all support.

Remark 7.3.1. *We also implement an* `Interpolation` *layer, corresponding to the piecewise linear interpolation of Vlastelica et al. (2020). However, to the best of our knowledge it cannot be cast as a probabilistic CO layer, so it does not support as many operations, and we only mention it for benchmarking purposes.*

We now present each row of Table 7.1 in more detail. The main goal of Sections 7.3.3.1, 7.3.3.2 and 7.3.3.3 is to explain how $\widehat{p}(\cdot|\theta)$ and $\widehat{f}(\theta)$ are computed, as well as their derivatives. We also draw connections with the regularization paradigm, which ease the introduction of Fenchel-Young losses in Section 7.5.2.3. A hasty reader can safely skip to Section 7.4.

7.3.3.1 Additive perturbation

A natural way to define a distribution on $\mathcal{V}$ is to solve (7.2) with a stochastic perturbation of the objective direction θ. Berthet et al. (2020) suggest the following additive perturbation mechanism:

$$\widehat{f}_\varepsilon^+(\theta) = \mathbb{E}\left[\underset{v \in \mathcal{V}}{\text{argmax}} (\theta + \varepsilon Z)^\top v\right] = \mathbb{E}\left[f(\theta + \varepsilon Z)\right] \tag{7.9}$$

```julia
using InferOpt, Zygote

p = compute_probability_distribution(layer, theta)
rand(p)
compute_expectation(p)

y = layer(theta)  # equal to the expectation of p
Zygote.jacobian(layer, theta)
```

Code sample 7.4: Supported operations for a probabilistic CO layer

where $\varepsilon > 0$ controls the amplitude of the perturbation, $Z \sim \mathcal{N}(0, I)$ is a standard Gaussian vector and the expectation is taken with respect to Z unless otherwise specified. Choosing ε is a trade-off between smoothness (large ε) and accuracy of the approximation (small ε). The associated probability distribution on $\mathcal{V}$ can be described explicitly:

$$\widehat{f}_\varepsilon^+(\theta) = \sum_{v \in \mathcal{V}} v \widehat{p}_\varepsilon^+(v|\theta) \qquad \text{with} \qquad \widehat{p}_\varepsilon^+(v|\theta) = \mathbb{P}\left(f(\theta + \varepsilon Z) = v\right). \tag{7.10}$$

Although the expectations cannot be expressed in closed form, they can be estimated using M Monte-Carlo samples $Z_1, ..., Z_M \sim \mathcal{N}(0, I)$. Increasing M yields smoother approximations but makes the complexity grow linearly.

The proofs of Proposition 7.3.2 and 7.3.3 are already given by Berthet et al. (2020), but we include them for comparison with the multiplicative case.

Proposition 7.3.2 (Differentiating through an additive perturbation (Berthet et al. 2020))**.** *We have:*

$$\nabla_\theta \widehat{p}_\varepsilon^+(v|\theta) = \frac{1}{\varepsilon}\mathbb{E}\left[\mathbb{1}\{f(\theta + \varepsilon Z) = v\} Z\right]$$
$$\mathrm{J}_\theta \widehat{f}_\varepsilon^+(\theta) = \frac{1}{\varepsilon}\mathbb{E}\left[f(\theta + \varepsilon Z) Z^\top\right]$$

Proof. In all of our proofs, the dominated convergence theorem is used implicitly to justify any differentiation under the integral sign. When dealing with polyhedral functions such as $\theta \longmapsto \max_{v \in \mathcal{V}} \theta^\top v$, we often write ∇_θ for simplicity even though they are only subdifferentiable, because the set of non-differentiability has measure zero.

The following change of variable is a diffeomorphism:

$$u = \theta + \varepsilon z \quad \iff \quad \frac{u - \theta}{\varepsilon} = z.$$

We apply it to the definition of $\widehat{p}_\varepsilon^+(v|\theta)$.

$$\begin{aligned}\widehat{p}_\varepsilon^+(v|\theta) &= \int_{\mathbb{R}^d} \mathbb{1}\{f(\theta + \varepsilon z) = v\}\ \nu(z)\ \mathrm{d}z \\ &= \int_{\mathbb{R}^d} \mathbb{1}\{f(u) = v\}\ \nu\left(\frac{u - \theta}{\varepsilon}\right) \frac{\mathrm{d}u}{\varepsilon^d}.\end{aligned}$$

We now differentiate with respect to θ before applying the reverse change of variable:

$$\begin{aligned}\nabla_\theta \widehat{p}_\varepsilon^+(v|\theta) &= \int_{\mathbb{R}^d} \mathbb{1}\{f(u) = v\} \left(\frac{-1}{\varepsilon}\nabla\nu\left(\frac{u-\theta}{\varepsilon}\right)\right)\frac{\mathrm{d}u}{\varepsilon^d} \\ &= \frac{-1}{\varepsilon}\int_{\mathbb{R}^d} \mathbb{1}\{f(\theta+\varepsilon z) = v\}\ \nabla\nu(z)\ \mathrm{d}z \\ &= \frac{1}{\varepsilon}\int_{\mathbb{R}^d} \mathbb{1}\{f(\theta+\varepsilon z) = v\}\ z\nu(z)\ \mathrm{d}z.\end{aligned}$$

The last equality holds because the standard Gaussian density satisfies $\nabla\nu(z) = -z\nu(z)$. From there, we deduce the Jacobian of $\widehat{f}_\varepsilon^+(\theta)$:

$$\begin{aligned}\mathrm{J}_\theta \widehat{f}_\varepsilon^+(\theta) &= \sum_{v\in\mathcal{V}} v\nabla_\theta \widehat{p}_\varepsilon^+(\theta, y)^\top \\ &= \frac{1}{\varepsilon}\int_{\mathbb{R}^d} \underbrace{\Big(\sum_{v\in\mathcal{V}} v\ \mathbb{1}\{f(\theta+\varepsilon z) = v\}\Big)}_{f(\theta+\varepsilon z)}\ z^\top\nu(z)\ \mathrm{d}z\end{aligned}$$

We arrive at the following simple expression, which was already given by Berthet et al. (2020):

$$\mathrm{J}_\theta \widehat{f}_\varepsilon^+(\theta) = \frac{1}{\varepsilon}\mathbb{E}\left[f(\theta+\varepsilon Z)Z^\top\right]$$

□

In order to recover the regularization associated with p_ε^+, we leverage convex conjugation. Let F_ε^+ be the function defined by

$$F_\varepsilon^+(\theta) = \mathbb{E}\left[\max_{v\in\mathcal{V}} (\theta+\varepsilon Z)^\top v\right]$$

and let $\Omega_\varepsilon^+ = (F_\varepsilon^+)^*$ denote its Fenchel conjugate.

Proposition 7.3.3 (Regularization associated with an additive perturbation (Berthet et al. 2020))**.** *The function Ω_ε^+ is convex, it satisfies $\mathrm{dom}(\Omega_\varepsilon^+) \subset \mathrm{conv}(\mathcal{V})$ and*

$$\widehat{f}_\varepsilon^+(\theta) = \operatorname*{argmax}_{\mu\in\mathrm{conv}(\mathcal{V})} \theta^\top\mu - \Omega_\varepsilon^+(\mu) = \widehat{f}_{\Omega_\varepsilon^+}(\theta).$$

Proof. Because $\Omega_\varepsilon^+ = (F_\varepsilon^+)^*$ is a Fenchel conjugate, it is automatically convex. Furthermore,

$$\begin{aligned}\Omega_\varepsilon^+(\mu) &= \sup_{\theta\in\mathbb{R}^d}\left\{\theta^\top\mu - F_\varepsilon^+(\theta)\right\} \\ &= \sup_{\theta\in\mathbb{R}^d}\left\{\theta^\top\mu - \mathbb{E}\left[\max_{v\in\mathcal{V}}(\theta+\varepsilon Z)^\top v\right]\right\}.\end{aligned}$$

We consider $\mu \notin \mathrm{conv}(\mathcal{V})$. By convex separation, there exists $\tilde{\theta} \in \mathbb{R}^d$ and $\alpha > 0$ such that $\tilde{\theta}^\top\mu \geq \alpha + \tilde{\theta}^\top v$ for all $v \in \mathcal{V}$. This implies that, for all $t > 0$,

$$\begin{aligned}\Omega_\varepsilon^+(\mu) &\geq t\tilde{\theta}^\top\mu - \mathbb{E}\left[\max_{v\in\mathcal{V}}\left(t\tilde{\theta}+\varepsilon Z\right)^\top v\right] \\ &\geq t\tilde{\theta}^\top\mu - t\mathbb{E}\left[\max_{v\in\mathcal{V}}\tilde{\theta}^\top v\right] - \varepsilon\mathbb{E}\left[\max_{v\in\mathcal{V}} Z^\top v\right] \\ &\geq t\alpha - \varepsilon\mathbb{E}\left[\max_{v\in\mathcal{V}} Z^\top v\right] \xrightarrow[t\to+\infty]{} +\infty\end{aligned}$$

We have shown that $\mu \notin \operatorname{dom}(\Omega_\varepsilon^+)$, and therefore $\operatorname{dom}(\Omega_\varepsilon^+) \subset \operatorname{conv}(\mathcal{V})$. We define

$$f_Z(\theta, v) = (\theta + \varepsilon Z)^\top v \qquad \text{so that} \qquad F_\varepsilon^+(\theta) = \mathbb{E}\left[\max_{v \in \mathcal{V}} f_Z(\theta, v)\right].$$

Danskin's theorem (Lemma A.3.1) helps us compute the gradient of F_ε^+:

$$\begin{aligned}
\nabla_\theta F_\varepsilon^+(\theta) &= \mathbb{E}\left[\nabla_\theta \left(\max_{v \in \mathcal{V}} f_Z(\theta, v)\right)\right] \\
&= \mathbb{E}\left[\nabla_1 f_Z\left(\theta, \operatorname*{argmax}_{v \in \mathcal{V}} f_Z(\theta, v)\right)\right] \\
&= \mathbb{E}\left[\operatorname*{argmax}_{v \in \mathcal{V}} f_Z(\theta, v)\right] \\
&= \mathbb{E}\left[\operatorname*{argmax}_{v \in \mathcal{V}} (\theta + \varepsilon Z)^\top v\right] = \widehat{f}_\varepsilon^+(\theta).
\end{aligned}$$

As shown by Berthet et al. (2020, Proposition 2.2), the function Ω_ε^+ is a Legendre type function, which means that

$$\nabla_\theta F_\varepsilon^+ = \nabla_\theta (\Omega_\varepsilon^+)^* = (\nabla_\theta \Omega_\varepsilon^+)^{-1}.$$

From this, we deduce

$$\begin{aligned}
\nabla_\theta F_\varepsilon^+(\theta) &= \operatorname*{argmax}_{\mu \in \mathbb{R}^d} \left\{\theta^\top \mu - \Omega_\varepsilon^+(\mu)\right\} \\
&= \operatorname*{argmax}_{\mu \in \operatorname{dom}(\Omega_\varepsilon^+)} \left\{\theta^\top \mu - \Omega_\varepsilon^+(\mu)\right\} = \widehat{f}_{\Omega_\varepsilon^+}(\theta).
\end{aligned}$$

Hence, we can conclude:

$$\widehat{f}_\varepsilon^+(\theta) = \nabla_\theta F_\varepsilon^+(\theta) = \operatorname*{argmax}_{\mu \in \operatorname{dom}(\Omega_\varepsilon^+)} \{\theta^\top \mu - \Omega_\varepsilon^+(\mu)\} = \widehat{f}_{\Omega_\varepsilon^+}(\theta).$$

To recover the formula given in Proposition 7.3.3, we simply remember that $\operatorname{dom}(\Omega_\varepsilon^+) \subseteq \operatorname{conv}(\mathcal{V})$. □

Code sample 7.5 shows how this translates into `InferOpt.jl` syntax.

7.3.3.2 Multiplicative perturbation

Since the Gaussian distribution puts mass on all of $\mathbb{R}^d$, it can happen that some components of $\theta + \varepsilon Z$ switch their sign with respect to θ. This may cause problems whenever the CO oracle for f has sign-dependent behavior. For instance, Dijkstra's algorithm for shortest paths requires all the edges of a graph to have a positive cost. In those cases, we need a sign-preserving kind of perturbation. Changing the distribution of Z to make it positive almost surely is not the right answer because it would bias the pipeline, leading to $\mathbb{E}[\theta + \varepsilon Z] > \theta$ for the componentwise order. So instead of being additive, the perturbation becomes multiplicative:

$$\widehat{f}_\varepsilon^{\odot}(\theta) = \mathbb{E}\left[\operatorname*{argmax}_{v \in \mathcal{V}} \left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right)^\top v\right] = \mathbb{E}\left[f\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right)\right] \tag{7.11}$$

Here, $\odot$ denotes the Hadamard product, and the exponential is taken componentwise. Since $\mathbb{E}[e^{\varepsilon Z}] = e^{\varepsilon^2 \mathbf{1}/2} \neq 1$, we add a correction term in the exponent to remove any bias: $\mathbb{E}[\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}] = \theta$. As before, the associated probability distribution is easy to describe:

$$\widehat{f}_\varepsilon^{\odot}(\theta) = \sum_{v \in \mathcal{V}} v \widehat{p}_\varepsilon^{\odot}(v|\theta) \qquad \text{with} \qquad \widehat{p}_\varepsilon^{\odot}(v|\theta) = \mathbb{P}\left(f\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) = v\right). \tag{7.12}$$

And Proposition 7.3.4 provides differentiation formulas that are very similar to the additive case.

Proposition 7.3.4 (Differentiating through a multiplicative perturbation)**.** *We have:*

$$\nabla_\theta \widehat{p}_\varepsilon^{\odot}(v|\theta) = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[\mathbb{1}\left\{f\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) = v\right\} Z\right]$$
$$\mathrm{J}_\theta \widehat{f}_\varepsilon^{\odot}(\theta) = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[f\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) Z^\top\right]$$

Proof. Suppose $\theta \in \mathbb{R}^d$ only has positive components. Then the following change of variable is a diffeomorphism:

$$u = \theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2} \qquad \Longleftrightarrow \qquad \frac{\log(u) - \log(\theta)}{\varepsilon} + \frac{\varepsilon \mathbf{1}}{2} = z$$

We apply it to the definition of $\widehat{p}_\varepsilon^{\odot}(v|\theta)$.

$$\begin{aligned}
\widehat{p}_\varepsilon^{\odot}(\theta, y) &= \int_{\mathbb{R}^d} \mathbb{1}\left\{f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right) = v\right\} \nu(z)\, \mathrm{d}z \\
&= \int_{(0,+\infty)^d} \mathbb{1}\{f(u) = v\}\, \nu\left(\frac{\log(u) - \log(\theta)}{\varepsilon} + \frac{\varepsilon \mathbf{1}}{2}\right) \frac{\mathrm{d}u}{\varepsilon^d \prod_i u_i}.
\end{aligned}$$

We now differentiate with respect to θ before applying the reverse change of variable:

$$\begin{aligned}
\nabla_\theta \widehat{p}_\varepsilon^{\odot}(\theta, y) &= \int_{(0,+\infty)^d} \mathbb{1}\{f(u) = v\} \left(\frac{-1}{\varepsilon\theta} \odot \nabla\nu\left(\frac{\log(u) - \log(\theta)}{\varepsilon} + \frac{\varepsilon \mathbf{1}}{2}\right)\right) \frac{\mathrm{d}u}{\varepsilon^d \prod_i u_i} \\
&= \frac{-1}{\varepsilon\theta} \odot \int_{\mathbb{R}^d} \mathbb{1}\left\{f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right) = v\right\} \nabla\nu(z)\, \mathrm{d}z \\
&= \frac{1}{\varepsilon\theta} \odot \int_{\mathbb{R}^d} \mathbb{1}\left\{f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right) = v\right\} z\nu(z)\, \mathrm{d}z.
\end{aligned}$$

From there, we deduce the Jacobian of $\widehat{f}_\varepsilon^{\odot}(\theta)$:

$$\begin{aligned}
\mathrm{J}_\theta \widehat{f}_\varepsilon^{\odot}(\theta) &= \sum_{v \in \mathcal{V}} v \nabla_\theta \widehat{p}_\varepsilon^{\odot}(\theta, y)^\top \\
&= \frac{1}{\varepsilon\theta} \odot \int_{\mathbb{R}^d} \underbrace{\left(\sum_{v \in \mathcal{V}} v\ \mathbb{1}\left\{f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right) = v\right\}\right)}_{f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right)} z^\top \nu(z)\, \mathrm{d}z
\end{aligned}$$

We arrive at a simple variant of the previous expression:

$$\mathrm{J}_\theta \widehat{f}_\varepsilon^{\odot}(\theta) = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[f\left(\theta \odot e^{\varepsilon z - \varepsilon^2 \mathbf{1}/2}\right) Z^\top\right]$$

□

As far as regularization is concerned, we need a slight tweak compared to the additive case. Let $F_\varepsilon^\odot$ be the function defined by

$$F_\varepsilon^\odot(\theta) = \mathbb{E}\left[\max_{v\in\mathcal{V}} \left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right)^\top v\right]$$

and let $\Omega_\varepsilon^\odot = (F_\varepsilon^\odot)^*$ denote its Fenchel conjugate. We define

$$\widehat{f}_\varepsilon^{\odot\text{scaled}}(\theta) = \mathbb{E}\left[e^{\varepsilon Z - \varepsilon^2\mathbf{1}/2} \odot f\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right)\right]$$

Proposition 7.3.5 (Regularization associated with a multiplicative perturbation)**.** *The function $\Omega_\varepsilon^\odot$ is convex and satisfies*

$$\widehat{f}_\varepsilon^{\odot\text{scaled}}(\theta) = \operatorname*{argmax}_{\mu\in\operatorname{dom}(\Omega_\varepsilon^\odot)} \theta^\top\mu - \Omega_\varepsilon^\odot(\mu) = \widehat{f}_{\Omega_\varepsilon^\odot}(\theta).$$

Unlike in the additive case, it is not $\widehat{f}_\varepsilon^\odot$ itself that can be viewed as the product of regularization with $\Omega_\varepsilon^\odot$. Furthermore, this time we have $\operatorname{dom}(\Omega_\varepsilon^\odot) \not\subseteq \operatorname{conv}(\mathcal{V})$.

Proof. Because $\Omega_\varepsilon^\odot = (F_\varepsilon^\odot)^*$ is a Fenchel conjugate, it is automatically convex. Furthermore,

$$\begin{aligned}
\Omega_\varepsilon^\odot(\mu) &= \sup_{\theta\in\mathbb{R}^d}\left\{\theta^\top\mu - F_\varepsilon^\odot(\theta)\right\} \\
&= \sup_{\theta\in\mathbb{R}^d}\left\{\theta^\top\mu - \mathbb{E}\left[\max_{v\in\mathcal{V}}\left(\theta\odot e^{\varepsilon Z-\varepsilon^2\mathbf{1}/2}\right)^\top v\right]\right\} \\
&= \sup_{\theta\in\mathbb{R}^d}\left\{\theta^\top\mu - \mathbb{E}\left[\max_{v\in\mathcal{V}}\theta^\top\left(v\odot e^{\varepsilon Z-\varepsilon^2\mathbf{1}/2}\right)\right]\right\}.
\end{aligned}$$

This last expression shows why we don't have $\operatorname{dom}(\Omega_\varepsilon^\odot) \subset \operatorname{conv}(\mathcal{V})$ (unlike in the additive case). Indeed, even when $\mu \notin \operatorname{conv}(\mathcal{V})$, the multiplicative scaling of v might allow it to compensate the inner product $\theta^\top\mu$ and stop $\Omega_\varepsilon^\odot(\mu)$ from going to $+\infty$. We define

$$f_Z(\theta, v) = \left(\theta \odot e^{\varepsilon Z - \varepsilon^2\mathbf{1}/2}\right)^\top v \qquad \text{so that} \qquad F_\varepsilon^\odot(\theta) = \mathbb{E}\left[\max_{v\in\mathcal{V}} f_Z(\theta, v)\right].$$

Danskin's theorem (Lemma A.3.1) helps us compute the gradient of $F_\varepsilon^\odot$:

$$\begin{aligned}
\nabla_\theta F_\varepsilon^\odot(\theta) &= \mathbb{E}\left[\nabla_\theta\left(\max_{v\in\mathcal{V}} f_Z(\theta, v)\right)\right] \\
&= \mathbb{E}\left[\nabla_1 f_Z\left(\theta, \operatorname*{argmax}_{v\in\mathcal{V}} f_Z(\theta, v)\right)\right] \\
&= \mathbb{E}\left[e^{\varepsilon Z-\varepsilon^2\mathbf{1}/2} \odot \operatorname*{argmax}_{v\in\mathcal{V}} f_Z(\theta, v)\right] \\
&= \mathbb{E}\left[e^{\varepsilon Z-\varepsilon^2\mathbf{1}/2} \odot \operatorname*{argmax}_{v\in\mathcal{V}}\left(\theta\odot e^{\varepsilon Z-\varepsilon^2\mathbf{1}/2}\right)^\top v\right] \\
&= \widehat{f}_\varepsilon^{\odot\text{scaled}}(\theta) \neq \widehat{f}_\varepsilon^\odot(\theta)
\end{aligned}$$

We could prove in a way similar to Berthet et al. (2020, Proposition 2.2) that $\Omega_\varepsilon^\odot$ is a Legendre type function, which means that

$$\nabla_\theta F_\varepsilon^\odot = \nabla_\theta (\Omega_\varepsilon^\odot)^* = (\nabla_\theta \Omega_\varepsilon^\odot)^{-1}.$$

From this, we deduce

$$\begin{aligned}\nabla_\theta F_\varepsilon^\odot(\theta) &= \operatorname*{argmax}_{\mu \in \mathbb{R}^d} \left\{\theta^\top \mu - \Omega_\varepsilon^\odot(\mu)\right\} \\ &= \operatorname*{argmax}_{\mu \in \operatorname{dom}(\Omega_\varepsilon^\odot)} \left\{\theta^\top \mu - \Omega_\varepsilon^\odot(\mu)\right\} = \widehat{f}_{\Omega_\varepsilon^\odot}(\theta).\end{aligned}$$

This time we cannot replace $\operatorname{dom}(\Omega_\varepsilon^\odot)$ by $\operatorname{conv}(\mathcal{V})$, but we still obtain a similar conclusion:

$$\widehat{f}_\varepsilon^{\odot\text{scaled}}(\theta) = \nabla_\theta F_\varepsilon^\odot(\theta) = \operatorname*{argmax}_{\mu \in \operatorname{dom}(\Omega_\varepsilon^\odot)} \{\theta^\top \mu - \Omega_\varepsilon^\odot(\mu)\} = \widehat{f}_{\Omega_\varepsilon^\odot}(\theta).$$

□

Code sample 7.5 shows how this translates into `InferOpt.jl` syntax.

7.3.3.3 Generic regularization

We now switch our focus to the case of an explicit regularization Ω. Provided the regularization is convex and smooth, approximate computation of $\widehat{f}_\Omega(\theta)$ is made possible by the Frank-Wolfe algorithm (Frank and Wolfe 1956). This algorithm is interesting for two reasons. First, it only requires access to the CO oracle f and the gradient of Ω. Second, its output is expressed as a convex combination of only a few polytope vertices (Jaggi 2013). In other words, the Frank-Wolfe algorithm does not just return a single point $\widehat{f}_\Omega(\theta) \in \operatorname{conv}(\mathcal{V})$: it also defines a sparse probability distribution $\widehat{p}_\Omega^{\text{FW}}(\cdot|\theta)$ over the vertices $\mathcal{V}$ such that

$$\widehat{f}_\Omega(\theta) = \sum_{v \in \mathcal{V}} v \widehat{p}_\Omega^{\text{FW}}(v|\theta).$$

This distribution is called sparse because most of the weights are actually zero. Note that $\widehat{p}_\Omega^{\text{FW}}(\cdot|\theta)$ is not uniquely specified by the regularization Ω, but instead depends on the precise implementation of the Frank-Wolfe algorithm (initialization, step size, convergence criterion, *etc.*). In particular, the number of atoms in the distribution is upper-bounded by the number of Frank-Wolfe iterations.

As pointed out by Blondel, Berthet, et al. (2022, Appendix C), there exists a function $g(p, \theta)$ defined on $\Delta^{\mathcal{V}} \times \mathbb{R}^d$ such that $\widehat{p}_\Omega^{\text{FW}}(\cdot|\theta)$ is a fixed point of its projected gradient operator $p \longmapsto \operatorname{proj}_{\Delta^{\mathcal{V}}}(p - \nabla_p g(p, \theta))$. Since the orthogonal projection onto the simplex $\Delta^{\mathcal{V}}$ is itself differentiable (Martins and Astudillo 2016), we can apply the implicit function theorem to this fixed point equation. Doing so yields gradients $\nabla_\theta \widehat{p}_\Omega(v|\theta)$ that we use to compute a Jacobian for $\widehat{f}_\Omega(\theta)$. Again, by sparsity, this sum only has a few non-zero terms, which makes it tractable:

$$\mathrm{J}_\theta \widehat{f}_\Omega(\theta) = \sum_{v \in \mathcal{V}} v \nabla_\theta \widehat{p}_\Omega^{\text{FW}}(v|\theta)^\top. \tag{7.13}$$

```julia
using InferOpt

perturbed_add = PerturbedAdditive(
    warcraft_maximizer;
    epsilon=0.5, nb_samples=10
)

perturbed_mult = PerturbedMultiplicative(
    warcraft_maximizer;
    epsilon=0.5, nb_samples=10
)
```

Code sample 7.5: Probabilistic CO layers defined by perturbation

```julia
using InferOpt

regularized = RegularizedGeneric(
    warcraft_maximizer;
    omega=y -> 0.5 * sum(y .^ 2),
    omega_grad=y -> y
)
```

Code sample 7.6: Probabilistic CO layer defined by regularization

Among all the possible functions Ω, the quadratic penalty $\Omega(\mu) = \frac{1}{2}\|\mu\|^2$ is particularly interesting. It gives rise to the SparseMAP method (Niculae et al. 2018), whose name comes from the sparsity of the Euclidean projection onto a polytope:

$$\widehat{f}_\Omega(\theta) = \underset{\mu \in \mathrm{conv}(\mathcal{V})}{\mathrm{argmax}} \left\{ \theta^\top \mu - \frac{1}{2}\|\mu\|^2 \right\} = \underset{\mu \in \mathrm{conv}(\mathcal{V})}{\mathrm{argmin}} \|\mu - \theta\|^2.$$

This is the one we used for the example of Code sample 7.6. Our implementation relies on the recent package `FrankWolfe.jl`[14] (Besançon, Carderera, and Pokutta 2022).

Remark 7.3.6. *Blondel, Martins, and Niculae (2020) also suggest distribution regularization, whereby $\Omega(\mu)$ is defined through a generalized entropy $H(p)$ on $\Delta^{\mathcal{V}}$:*

$$\Omega(\mu) = -\max_{p \in \Delta^{\mathcal{V}}} H(p) \quad s.t. \quad \mathbb{E}_p[V] = \mu.$$

Distribution regularization can only be computed explicitly for certain entropies H (Shannon entropy, Gini index) and certain polytopes $\mathrm{conv}(\mathcal{V})$ (unit simplex, spanning trees, etc.). In each case, a custom combinatorial algorithm is required. Since we aim for a generic approach, we only consider mean regularization, which is defined directly on the expectation μ.

7.3.4 The case of inexact CO oracles

In our discussion so far, an implicit assumption was that the CO oracle f returns an exact solution to Equation (7.2). For most polynomial problems (as well as some NP-hard problems which are tractable in practice), this is perfectly reasonable. But in some cases, exact solutions are too expensive to compute. Then, our CO oracle may only be able to return an inexact solution, for instance because branch & bound has to be interrupted before the whole tree can be explored. What kind of impact does this have on the precision of the computed Jacobian?

Let us denote by f a hypothetical exact oracle, and by g an inexact oracle.

[14] https://github.com/ZIB-IOL/FrankWolfe.jl

Proposition 7.3.7 (Jacobian precision for inexact oracles – perturbed case)**.** *Suppose we use g instead of f with additive (resp. multiplicative) perturbation. Then the error on the Jacobian of the probabilistic CO layer satisfies:*

$$\left\|\mathrm{J}_\theta \widehat{g}_\varepsilon^{+}(\theta) - \mathrm{J}_\theta \widehat{f}_\varepsilon^{+}(\theta)\right\|^2 \leq \frac{\sqrt{d}}{\varepsilon}\|g - f\|_\infty$$
$$\left\|\mathrm{J}_\theta \widehat{g}_\varepsilon^{\odot}(\theta) - \mathrm{J}_\theta \widehat{f}_\varepsilon^{\odot}(\theta)\right\|^2 \leq \frac{\sqrt{d}}{\varepsilon \min_i|\theta_i|}\|g - f\|_\infty.$$

While requiring the inexact oracle g to be uniformly close to f is quite restrictive, this result does provide heuristic justification for the use of inexact oracles in practice.

Proof. We start with additive perturbation. By Proposition 7.3.2, we have:

$$\begin{aligned}\mathrm{J}_\theta \widehat{g}_\varepsilon^{+}(\theta) - \mathrm{J}_\theta \widehat{f}_\varepsilon^{+}(\theta) &= \frac{1}{\varepsilon}\mathbb{E}\left[g(\theta + \varepsilon Z)Z^\top\right] - \frac{1}{\varepsilon}\mathbb{E}\left[f(\theta + \varepsilon Z)Z^\top\right] \\ &= \frac{1}{\varepsilon}\mathbb{E}\left[\left(g(\theta + \varepsilon Z) - f(\theta + \varepsilon Z)\right)Z^\top\right].\end{aligned}$$

We bound the spectral norm of the error using Jensen's inequality:

$$\begin{aligned}\left\|\mathrm{J}_\theta \widehat{g}_\varepsilon^{+}(\theta) - \mathrm{J}_\theta \widehat{f}_\varepsilon^{+}(\theta)\right\|^2 &\leq \frac{1}{\varepsilon^2}\mathbb{E}\left\|\left(g(\theta + \varepsilon Z) - f(\theta + \varepsilon Z)\right)Z^\top\right\|^2 \\ &= \frac{1}{\varepsilon^2}\mathbb{E}\left[\|g(\theta + \varepsilon Z) - f(\theta + \varepsilon Z)\|^2\|Z\|^2\right] \\ &\leq \frac{1}{\varepsilon^2}\|g - f\|_\infty^2\mathbb{E}\left[\|Z\|^2\right] \\ &= \frac{d}{\varepsilon^2}\|g - f\|_\infty^2.\end{aligned}$$

We now move on to multiplicative perturbation. For multiplicative perturbation, following the same proof starting from Proposition 7.3.4 yields a similar inequality:

$$\left\|\mathrm{J}_\theta \widehat{g}_\varepsilon^{\odot}(\theta) - \mathrm{J}_\theta \widehat{f}_\varepsilon^{\odot}(\theta)\right\|^2 \leq \frac{d}{\varepsilon^2 \min_i|\theta_i|^2}\|g - f\|_\infty^2.$$

□

As for generic regularization, Jacobian precision guarantees are given by Theorem 1 of Blondel, Berthet, et al. (2022).

7.4 Learning by experience

Now that we have seen several ways to construct probabilistic CO layers, we turn to the definition of an appropriate loss function. Let us start with learning by experience, which takes place when we only have access to input samples without target outputs. In that case, Equation (7.5) simplifies as

$$\min_w \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\Big(f\big(\varphi_w(x^{(i)})\big)\Big). \tag{7.14}$$

As we will see below, the *regret*, which is the natural choice of loss, does not yield interesting gradients. That is why we propose a family of *smooth regret surrogates* derived from our probabilistic CO layers, and explain how to differentiate them. While similar losses have been hinted at in previous works, to the best of our knowledge, our general point of view is new.

To make notations lighter, we restrict ourselves to a single input x. Furthermore, we write losses as functions of θ instead of w. Indeed, our losses do not just rely on $y = f(\theta)$: they use f as an ingredient internally. In practice, we leave it to AD to exploit the relation $\theta = \varphi_w(x)$ in order to compute gradients with respect to w.

7.4.1 Minimizing a smooth regret surrogate

When we learn by experience, the problem statement usually includes a cost function $c : \mathcal{V} \to \mathbb{R}$, and we want our pipeline to generate solutions that are as cheap as possible. Internally, this cost function may use parameters that are unknown to us at prediction time: typically, it may assess the quality of our solution using the true objective direction $\bar{\theta}$. It may be useful to think about c as the feedback provided by an outside evaluator, rather than a function we implement ourselves.

The natural loss to minimize is the cost incurred by our prediction pipeline, also called regret:

$$\mathcal{R}(\theta) = c(f(\theta)). \tag{7.15}$$

This function relies on the CO oracle f, which is piecewise constant. Our spontaneous impulse would be to replace the CO oracle f with a probabilistic CO layer $\widehat{f}$, thus minimizing $c(\widehat{f}(\theta))$. Unfortunately, the cost function c is not necessarily smooth either. To make matters worse, c may only be defined on vertices $v \in \mathcal{V}$, and not on general convex combinations $\mu \in \operatorname{conv}(\mathcal{V})$.

The solution we propose relies on the *pushforward measure* (also called image measure) of $\widehat{p}(\cdot|\theta)$ with respect to the function c. Recall that a probabilistic CO layer is defined by $\widehat{f}(\theta) = \mathbb{E}_{\widehat{p}(\cdot|\theta)}[V]$. To compose it with an arbitrary cost, instead of applying c outside the expectation, we apply it inside the expectation. In other words, we first push the measure $\widehat{p}(\cdot|\theta)$ forward through the function c, before taking the expectation. This gives rise to the notion of *expected regret*:

$$\mathcal{R}_{\widehat{p}}(\theta) = \mathbb{E}_{\widehat{p}(\cdot|\theta)}[c(V)] \tag{7.16}$$

By integration, this loss is just as smooth as the probability mapping $\theta \longmapsto \widehat{p}(\cdot|\theta)$, which means we can compute its gradient easily. We therefore suggest using the expected regret $\mathcal{R}_{\widehat{p}}$ that stems from the probabilistic CO layers $\widehat{p}^{+}_{\varepsilon}$, $\widehat{p}^{\odot}_{\varepsilon}$, and $\widehat{p}^{\mathrm{FW}}_{\Omega}$ defined in Section 7.3.

Note that if c is linear and defined on all of $\operatorname{conv}(\mathcal{V})$, then $\mathbb{E}_{\widehat{p}(\cdot|\theta)}[c(V)] = c\left(\mathbb{E}_{\widehat{p}(\cdot|\theta)}[V]\right)$ and the two quantities coincide. Furthermore, if c is convex, then $\mathbb{E}_{\widehat{p}(\cdot|\theta)}[c(V)] \geq c\left(\mathbb{E}_{\widehat{p}(\cdot|\theta)}[V]\right)$ by Jensen's inequality, which means the expected regret is an upper bound.

Code sample 7.7 demonstrates how to define an expected regret from probabilistic CO layers, while Code sample 7.8 shows that we can compute and differentiate it automatically. Finally, Code sample 7.9 displays a complete program for learning by experience. The rest of this section explains how to compute derivatives of the expected regret $\mathcal{R}_{\widehat{p}}$ and can be skipped without danger.

Remark 7.4.1. *Since the learning problem is non-convex, we may also try to minimize the (non-smooth) regret $\mathcal{R}$ using global optimization algorithms such as DIRECT (Jones, Perttunen, and Stuckman 1993). Perhaps surprisingly, this has been shown to yield good results when φ_w is a generalized linear model and the dimension of the weights w is not too large, i.e., non greater that* 100 *(Parmentier 2021a). When the ML layer φ_w is a large neural network, we cannot use this approach anymore.*

```julia
using InferOpt

regret_pert = Pushforward(
    perturbed_add, warcraft_cost
)
regret_reg = Pushforward(
    regularized, warcraft_cost
)
```

Code sample 7.7: Expected regrets associated with probabilistic CO layers

```julia
using Zygote

R = regret(theta)
Zygote.gradient(regret, theta)
```

Code sample 7.8: Supported operations for an expected regret

```julia
using Flux, InferOpt

gradient_optimizer = ADAM()
parameters = Flux.params(warcraft_encoder)
data = images

function pipeline_loss(x)
    theta = warcraft_encoder(x)
    return regret(theta)
end

for epoch in 1:1000
    train!(pipeline_loss, parameters, data, gradient_optimizer)
end
```

Code sample 7.9: Learning with an expected regret

7.4.2 Derivatives of the regret for learning by experience

When $\widehat{p}$ comes from a random perturbation, we can formulate the gradient of the expected regret as an expectation too, and approximate it with Monte-Carlo samples.

Proposition 7.4.2 (Gradient of the expected regret, perturbation setting)**.** *We have:*

$$\nabla_\theta \mathcal{R}_{\widehat{p}^+_\varepsilon}(\theta) = \frac{1}{\varepsilon}\mathbb{E}\left[(c \circ f)(\theta + \varepsilon Z)Z\right]$$
$$\nabla_\theta \mathcal{R}_{\widehat{p}^\odot_\varepsilon}(\theta) = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[(c \circ f)\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) Z\right].$$

Proof. For the additive perturbation, it is a consequence of Proposition 7.3.2. For the multiplicative perturbation, it is a consequence of Proposition 7.3.4. □

These gradients obey a simple logic: the more the perturbation Z increases the cost of a solution, the more positive weight it gets, and vice versa. To see it, we remember that $\mathbb{E}[Z] = 0$ for a standard Gaussian, and rewrite the regret gradients as follows:

$$\nabla_\theta \mathcal{R}_{\widehat{p}^+_\varepsilon}(\theta) = \frac{1}{\varepsilon}\mathbb{E}\left[(c \circ f)(\theta + \varepsilon Z)Z - (c \circ f)(\theta)Z\right]$$
$$\nabla_\theta \mathcal{R}_{\widehat{p}^\odot_\varepsilon}(\theta) = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[(c \circ f)\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) Z - (c \circ f)(\theta)Z\right].$$

On the other hand, when $\widehat{p}$ is derived from an explicit regularization Ω, the expected regret is amenable to implicit differentiation of the Frank-Wolfe algorithm. Once more, the sparsity property makes exact computation tractable by reducing the number of terms in the sum:

$$\nabla_\theta \mathcal{R}_{\widehat{p}^{\mathrm{FW}}_\Omega}(\theta) = \sum_{v \in \mathcal{V}} c(v) \nabla_\theta \widehat{p}^{\mathrm{FW}}_\Omega(v|\theta)$$

Remark 7.4.3. *Although the previous discussion focuses on a scalar-valued cost, it actually applies to any pushforward function c, even with vector values. The formulas for the generic Jacobian are given below:*

$$\mathrm{J}_\theta \mathbb{E}_{\widehat{p}^+_\varepsilon(\cdot|\theta)}[c(V)] = \frac{1}{\varepsilon}\mathbb{E}\left[(c \circ f)(\theta + \varepsilon Z)Z^\top\right]$$
$$\mathrm{J}_\theta \mathbb{E}_{\widehat{p}^\odot_\varepsilon(\cdot|\theta)}[c(V)] = \frac{1}{\varepsilon\theta} \odot \mathbb{E}\left[(c \circ f)\left(\theta \odot e^{\varepsilon Z - \varepsilon^2 \mathbf{1}/2}\right) Z^\top\right]$$
$$\mathrm{J}_\theta \mathbb{E}_{\widehat{p}^{FW}_\Omega(\cdot|\theta)}[c(V)] = \sum_{v \in \mathcal{V}} c(v) \nabla_\theta \widehat{p}^{FW}_\Omega(v|\theta)^\top$$

At the moment, `InferOpt.jl` *only handles the case where c is a fully-defined function without free parameters. In the near future, we will add support for the case where c is itself an ML layer with learnable weights.*

7.5 Learning by imitation

We now move on to learning by imitation, where additional information is used to guide the training procedure. For each input sample $x^{(i)}$, we assume we have access to a *target* $\bar{t}^{(i)}$. In that case,

Equation (7.5) simplifies as

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\Big(f\big(\varphi_w(x^{(i)})\big), \bar{t}^{(i)}\Big), \tag{7.17}$$

and we can see that the loss takes the target as an additional argument. In this section, we introduce imitation losses that are well-suited to hybrid ML-CO pipelines (similar to the cost-sensitive Fenchel-Young losses of Blondel, Martins, and Niculae (2020)), and explain how to compute their gradients. As in Section 7.4, we only consider a single input x, and we write losses as $\mathcal{L}(\theta, \bar{t})$.

7.5.1 A loss that takes the optimization layer into account

There are two main kinds of target. The first one is a good quality solution $\bar{t} = \bar{y}$. The second one is the true objective direction $\bar{\theta}$, from which we can also deduce $\bar{y} = f(\bar{\theta})$, so that $\bar{t} = (\bar{\theta}, \bar{y})$. When learning by imitation, it is tempting to focus only on reproducing the targets, but this would be misguided. To explain why, we revisit the pipeline of Equation (7.4).

Remember that we may have access to the true objective direction $\bar{\theta}$ during training, but at prediction time, the CO oracle f is applied to the encoder output $\theta = \varphi_w(x)$ instead. Minimizing a naive square loss like $\|\varphi_w(x) - \bar{\theta}\|^2$ completely neglects the asymmetric impacts of the prediction errors on θ: for example, overestimating or underestimating θ may have very different consequences on the quality of the downstream solution. That is why, according to Elmachtoub and Grigas (2022), we need a loss function that takes the optimization step into account. The same holds true when we have access to a precomputed solution $\bar{y}$. Berthet et al. (2020) present experiments showing that the naive square loss $\|\widehat{f}(\varphi_w(x)) - \bar{y}\|^2$ performs poorly compared with more refined approaches. Our own numerical findings (Section 7.6) support their conclusion.

To sum up, we want a loss that does not neglect the optimization step. Let y temporarily denote the output of our pipeline. When surveying the literature, we realized that most flavors of imitation learning use losses that combine the same components:

$$\mathcal{L}^{\text{aux}}(\theta, \bar{t}, y) = \underbrace{\ell(y, \bar{t})}_{\text{base loss}} + \underbrace{\theta^\top (y - \bar{y})}_{\substack{\text{gap between } y \text{ and } \bar{y} \text{ for the} \\ \text{CO problem (7.2)}}} - \underbrace{(\Omega(y) - \Omega(\bar{y}))}_{\text{regularization term}} \tag{7.18}$$

Here is another way to write it:

$$\mathcal{L}^{\text{aux}}(\theta, \bar{t}, y) = \underbrace{\ell(y, \bar{t})}_{\text{base loss}} + \underbrace{\Big(\theta^\top y - \Omega(y)\Big) - \Big(\theta^\top \bar{y} - \Omega(\bar{y})\Big)}_{\substack{\text{gap between } y \text{ and } \bar{y} \\ \text{for the regularized CO problem (7.8)}}}$$

The base loss $\ell(y, \bar{t})$ is similar in spirit to the cost function $c(y)$ from Section 7.4. But it is the gap term that truly makes it possible for the optimization problem to play a role in the loss. Indeed, minimizing the gap encourages the (regularized) CO problem to output a solution y that is close to the target $\bar{y}$.

Putting these components together yields a linear function of θ, and we can remove the dependency in y by maximizing over y:

$$\mathcal{L}^{\text{gen}}(\theta, \bar{t}) = \max_{y \in \text{dom}(\Omega)} \mathcal{L}^{\text{aux}}(\theta, \bar{t}, y) = \max_{y \in \text{dom}(\Omega)} \Big[\ell(y, \bar{t}) + \theta^\top (y - \bar{y}) - (\Omega(y) - \Omega(\bar{y}))\Big]. \tag{7.19}$$

The following result justifies why this is an interesting loss.

Method	Notation	Target	Base loss	Regul.	Loss formula
S-SVM	$\mathcal{L}_\ell^{\text{S-SVM}}$	$\bar{y}$	$\ell(y, \bar{y})$	No	$\max_y \ell(y, \bar{y}) + \theta^\top (y - \bar{y})$
SPO+	$\mathcal{L}^{\text{SPO+}}$	$(\bar{\theta}, \bar{y})$	$\bar{\theta}^\top (\bar{y} - y)$	No	$\max_y \bar{\theta}^\top (\bar{y} - y) + 2\theta^\top (y - \bar{y})$
FY	$\mathcal{L}_\Omega^{\text{FY}}$	$\bar{y}$	0	Yes	$\max_y \theta^\top (y - \bar{y}) - (\Omega(y) - \Omega(\bar{y}))$
Generic	$\mathcal{L}^{\text{gen}}$	$\bar{t}$	$\ell(y, \bar{t})$	Yes	$\max_y \ell(y, \bar{t}) + \theta^\top (y - \bar{y}) - (\Omega(y) - \Omega(\bar{y}))$

Table 7.2: A common decomposition for loss functions in imitation learning

Proposition 7.5.1 (Properties of the generic loss for learning by imitation)**.** *The function $\mathcal{L}^{gen}(\theta, \bar{t})$ is convex with respect to θ, and a subgradient is given by*

$$\left(\underset{y \in \text{dom}(\Omega)}{\text{argmax}} \, \mathcal{L}^{aux}(\theta, \bar{t}, y) \right) - \bar{y} \quad \in \quad \partial_\theta \mathcal{L}^{gen}(\theta, \bar{t}). \tag{7.20}$$

Proof. As a pointwise maximum of affine functions, $\theta \longmapsto \mathcal{L}(\theta, \bar{t})$ is convex. Its subgradient is obtained using Danskin's theorem (Lemma A.3.1). □

The idea is that solving $\text{argmax}_{y \in \text{dom}(\Omega)} \mathcal{L}^{\text{aux}}(\theta, \bar{t}, y)$ should not be much harder than the regularized CO problem (7.8). Therefore, using such a loss function dispenses us from differentiating through the probabilistic CO layer: most of the time, we only need to compute the layer output in order to obtain a loss subgradient for free.

Our formulation is slightly different from the cost-sensitive Fenchel-Young losses of Blondel, Martins, and Niculae (2020) because the cost $\ell(y, \bar{t})$ can include generic targets such as $\bar{\theta}$, making it a generalization of SPO+ as well (see below).

7.5.2 Collection of losses for learning by imitation

Several prominent loss functions from the literature are special cases of our decomposition (7.19): we gather them in Table 7.2. Code sample 7.10 clarifies their construction, while Code sample 7.11 displays supported operations. Finally, the entire program necessary for learning by imitation is shown on Code sample 7.12.

In Sections 7.5.2.1, 7.5.2.2, 7.5.2.3 and 7.5.2.4, we go over these special cases to explain how to compute each loss and its subgradient using Equation (7.20). They can be skipped without danger.

Remark 7.5.2. *While the S-SVM and SPO+ losses do not fall within the framework of probabilistic CO layers (due to the absence of regularization), we still include them for benchmarking purposes.*

7.5.2.1 Structured support vector machines

The S-SVM was among the first methods introduced for learning in structured spaces (Tsochantaridis et al. 2005). Given a target solution $\bar{y}$ and an underlying distance function $\ell(y, \bar{y})$ on $\mathcal{V}$, the S-SVM loss is computed as follows:

$$\mathcal{L}_\ell^{\text{S-SVM}}(\theta, \bar{y}) = \max_{y \in \mathcal{V}} \{\ell(y, \bar{y}) + \theta^\top (y - \bar{y})\}. \tag{7.21}$$

```julia
using InferOpt

fyl_pert = FenchelYoungLoss(perturbed_add)
fyl_reg = FenchelYoungLoss(regularized)
spol = SPOPlusLoss(warcraft_maximizer)
```

Code sample 7.10: Example imitation losses

```julia
using Zygote

L = loss(theta, y_ref)
Zygote.gradient(loss, theta, y_ref)
```

Code sample 7.11: Supported operations for an imitation loss

```julia
using Flux, InferOpt

gradient_optimizer = ADAM()
parameters = Flux.params(warcraft_encoder)
data = zip(images, paths)

function pipeline_loss(x, y)
    theta = warcraft_encoder(x)
    return loss(theta, y)
end

for epoch in 1:1000
    train!(pipeline_loss, parameters, data, gradient_optimizer)
end
```

Code sample 7.12: Learning with an imitation loss

The subgradient formula (7.20) becomes

$$\operatorname*{argmax}_{y \in \mathcal{V}} \{\ell(y, \bar{y}) + \theta^\top (y - \bar{y})\} - \bar{y} \quad \in \quad \partial_\theta \mathcal{L}_\ell^{\text{S-SVM}}.$$

Note that due to the presence of ℓ, computing a subgradient requires an auxiliary solver that is different from the linear oracle f. This is why we do not illustrate the S-SVM with a code sample. In `InferOpt.jl`, we only implement this auxiliary solver for the unit simplex, in the case where ℓ is the Hamming distance. However, we also provide a generic layer where the user can plug in the relevant auxiliary solver.

7.5.2.2 Smart "predict, then optimize"

The SPO paradigm is applicable when the true objective direction $\bar{\theta}$ is known (remember that in this case, we have $\bar{y} = f(\bar{\theta})$). Elmachtoub and Grigas (2022) define the SPO+ loss function as follows:

$$\begin{aligned} \mathcal{L}^{\text{SPO}+}(\theta, \bar{\theta}) &= (2\theta - \bar{\theta})^\top f(2\theta - \bar{\theta}) + (\bar{\theta} - 2\theta)^\top \bar{y} \\ &= \max_{y \in \mathcal{V}} \left\{ \bar{\theta}^\top (\bar{y} - y) + 2\theta^\top (y - \bar{y}) \right\}. \end{aligned} \tag{7.22}$$

It can be seen as a special case of S-SVM. But this time, computing the loss and its subgradient only requires calling f twice:

$$2f(2\theta - \bar{\theta}) - 2\bar{y} \quad \in \quad \partial_\theta \mathcal{L}^{\text{SPO}+}(\theta, \bar{\theta}).$$

7.5.2.3 Fenchel-Young losses

The framework of Fenchel-Young losses is built on the theory of convex conjugates, in particular the Fenchel-Young inequality (Blondel, Martins, and Niculae 2020). Starting from a target solution $\bar{y}$ and a regularization Ω, a loss is constructed as follows:

$$\begin{aligned} \mathcal{L}_\Omega^{\mathrm{FY}}(\theta, \bar{y}) &= \Omega^*(\theta) + \Omega(\bar{y}) - \theta^\top \bar{y} \\ &= \max_{y \in \mathrm{conv}(\mathcal{V})} \Big(\theta^\top y - \Omega(y)\Big) - \Big(\theta^\top \bar{y} - \Omega(\bar{y})\Big) \end{aligned} \tag{7.23}$$

This time, the loss and subgradient require access to $\widehat{f}_\Omega$:

$$\widehat{f}_\Omega(\theta) - \bar{y} \quad \in \quad \partial_\theta \mathcal{L}_\Omega^{\mathrm{FY}}(\theta, \bar{y}).$$

As can be inferred from the expression above, there are deep connections between Fenchel-Young losses and the regularization paradigm of Section 7.3.2. In particular, it is also possible to use implicit regularization by perturbation (Berthet et al. 2020). The fact that we cannot compute $\Omega_\varepsilon^+(y)$ or $\Omega_\varepsilon^\odot(y)$ is not a real obstacle: since those terms do not depend on θ, we can just drop them from the loss during training. We end up with the following estimators for the loss and its subgradient:

$$\mathcal{L}_{\Omega_\varepsilon^+}^{\mathrm{FY}}(\theta, \bar{y}) = F_\varepsilon^+(\theta) - \theta^\top \bar{y} \qquad \widehat{f}_\varepsilon^+(\theta) - \bar{y} \in \partial_\theta \mathcal{L}_{\Omega_\varepsilon^+}^{\mathrm{FY}}(\theta, \bar{y})$$
$$\mathcal{L}_{\Omega_\varepsilon^\odot}^{\mathrm{FY}}(\theta, \bar{y}) = F_\varepsilon^\odot(\theta) - \theta^\top \bar{y} \qquad \widehat{f}_\varepsilon^{\odot\mathrm{scaled}}(\theta) - \bar{y} \in \partial_\theta \mathcal{L}_{\Omega_\varepsilon^\odot}^{\mathrm{FY}}(\theta, \bar{y})$$

7.5.2.4 Generic imitation loss

Of course, it is tempting to fill in the blanks of Table 7.2 by combining every single term of the loss decomposition (7.19). To the best of our knowledge, this has not been done before in the literature, but there is no theoretical obstacle.

If we use this generic loss together with regularization, then it is interesting to remark that $\mathcal{L}(\theta, \bar{t})$ acts as a convex upper bound on the base loss $\ell(\widehat{f}_\Omega(\theta), \bar{t})$. Indeed, since $\bar{y}$ is a worse solution than $\widehat{f}_\Omega(\theta)$ for (7.8), we have

$$\begin{aligned} \ell(\widehat{f}_\Omega(\theta), \bar{t}) &\leq \ell(\widehat{f}_\Omega(\theta), \bar{t}) + \Big[\theta^\top \widehat{f}_\Omega(\theta) - \Omega(\widehat{f}_\Omega(\theta))\Big] - \Big[\theta^\top \bar{y} - \Omega(\bar{y})\Big] \\ &\leq \max_{y \in \mathrm{conv}(\mathcal{V})} \Big(\ell(y, \bar{t}) + \theta^\top (y - \bar{y}) - (\Omega(y) - \Omega(\bar{y}))\Big) = \mathcal{L}(\theta, \bar{t}). \end{aligned}$$

Therefore, our generic loss can be seen as a crossover between the Fenchel-Young loss and a problem-specific base loss. It is not yet implemented in `InferOpt.jl`, and we leave its thorough testing for future work.

7.6 Numerical experiments

In this first version of the paper, we only present one application of our `InferOpt.jl` package: Warcraft shortest paths. Experiments on other concrete problems will be added to the final version.

Hyperparameter	Description
`epsilon`	Scale of the noise for perturbation.
`nb_samples`	Number of noise samples M for perturbation.
`batch_size`	Size of the batches to compute gradients.
`lr_start`	Starting learning rate.

Table 7.3: Hyperparameters for learning Warcraft shortest paths.

7.6.1 Shortest paths on Warcraft maps

We come back to our guiding example of Section 7.2.3. Our aim is to illustrate the various learning settings introduced in this paper, and to evaluate their relative performance. We do so with two kinds of shortest path (SP) oracles. The first one uses Dijkstra's algorithm. The second one uses the Ford-Bellman algorithm with a bounded number of iterations.

7.6.1.1 Experimental setting

In every experiment presented here, we only consider a sub-dataset, made up of 1% of the original Warcraft dataset from Vlastelica et al. (2020). It contains 200 samples, which we split into 80 training samples, 100 validation samples (for hyperparameter tuning) and 20 test samples (for performance evaluation). The train, test and validation sets are the same for each learning setting. Our motivation for reducing the dataset is to show that we can still obtain convincing results with a limited amount of computation.

We use the `Metalhead.jl` package to build a truncated ResNet18 CNN, `Flux.jl` to train our pipelines with the Adam optimizer (Kingma and Ba 2015), and `GridGraphs.jl` to compute shortest paths. Our code will soon be made available in the `WarcraftShortestPaths.jl`[15] repository (which is still private at the time of writing). For each learning setting, we tune a subset of the hyperparameters stated in Table 7.3. The experiments are conducted on a MacBook Pro with 2,3 GHz Intel Core i9, 8 cores and 16 Go 2667 MHz DDR4 RAM.

To obtain a precise learning setting, we need to define:

1. The combinatorial problem we need to solve, along with an appropriate oracle.
2. The probabilistic CO layer used to wrap said oracle.
3. The data we have at our disposal to train our pipeline.
4. The loss function we want to minimize.

No matter the setting, the data always contains a list of RGB map images. When we learn by experience, we also have access to a black box cost function, which evaluates paths based on the true cell costs (see the beginning of Section 7.4). On the other hand, when we learn by imitation, we add targets to the maps (as defined in Section 7.5). The target in our case always includes the optimal path, with or without the true cell costs.

All those ingredients are detailed in Table 7.4 for each setting we consider. The names of the first column are reused in the legends of Figure 7.3. Most of the probabilistic CO layers considered

[15] https://github.com/LouisBouvier/WarcraftShortestPaths.jl

Name	CO problem (CO oracle)	Probabilistic CO layer	Exp./Imit. Target	Loss
Cost perturbed multiplicative noise	SP with non-negative costs (Dijkstra)	Multiplicative perturbation	Experience No target	Perturbed cost
Cost perturbed additive noise	SP on an extended acyclic graph (Ford-Bellman)	Additive perturbation	Experience No target	Perturbed cost
Cost regularized half square norm	SP on an extended acyclic graph (Ford-Bellman)	Half square norm	Experience No target	Regularized cost
SPO+	SP on an extended acyclic graph (Ford-Bellman)	No regularization	Imitation Cost and path	SPO+ loss
MSE perturbed multiplicative noise	SP with non-negative costs (Dijkstra)	Multiplicative perturbation	Imitation Path	Mean squared error
MSE regularized half square norm	SP on an extended acyclic graph (Ford-Bellman)	Half square norm	Imitation Path	Mean squared error
Fenchel-Young perturbed multiplicative noise	SP with non-negative costs (Dijkstra)	Multiplicative perturbation	Imitation Path	Fenchel-Young
Fenchel-Young perturbed additive noise	SP on an extended acyclic graph (Ford-Bellman)	Additive perturbation	Imitation Path	Fenchel-Young
Fenchel-Young regularized half square norm	SP on an extended acyclic graph (Ford-Bellman)	Half square norm	Imitation Path	Fenchel-Young

Table 7.4: Learning settings for Warcraft shortest paths.

in this paper do not prevent the objective vector θ from changing its sign, and the same goes for the losses. As a result, we need oracles able to accommodate negative cell costs. That is why we use the Ford-Bellman algorithm, while limiting the number of iterations to the number of nodes in the grid graph (to ensure termination even with negative cycles). Our multiplicative perturbation is the only approach that preserves non-negative costs. It enables us to apply Dijkstra's algorithm, which has a smaller time and space complexity.

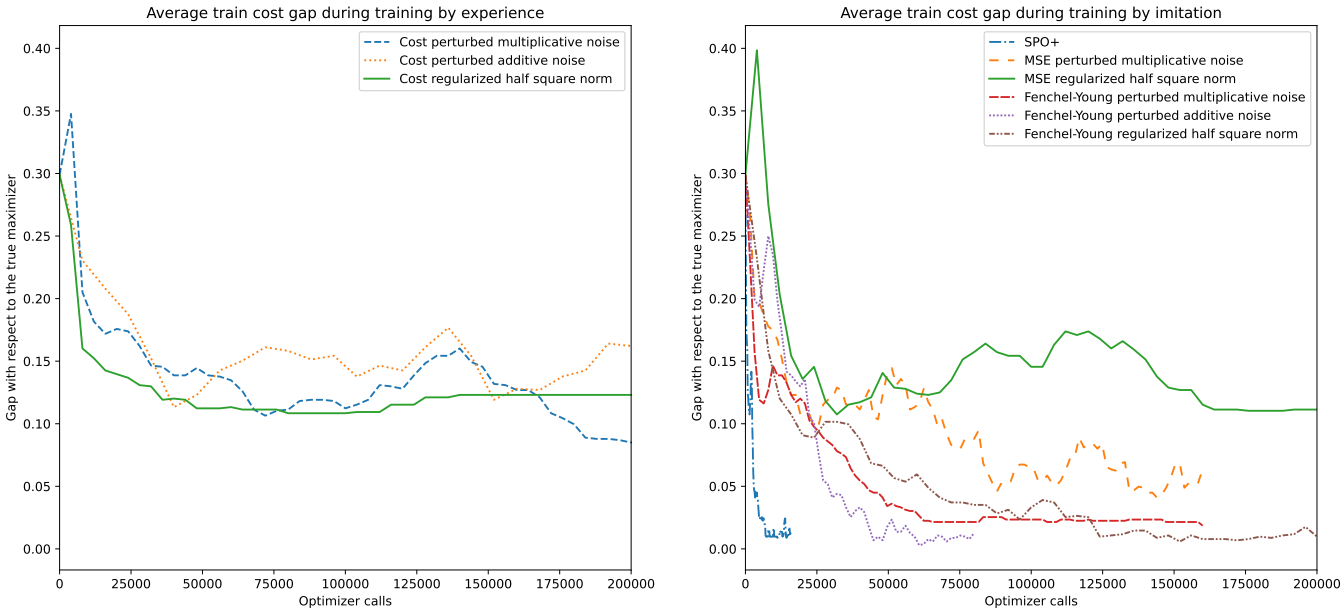
7.6.1.2 Results

In Figure 7.3, we show the average train (Figure 7.3a) and test (Figure 7.3b) optimality gaps, computed using the true cell costs. We compare all the settings detailed in Table 7.4, except MSE base loss + additive noise (we could not get satisfactory results using only 80 training samples). To quantify training effort, instead of counting epochs (*i.e.*, passes through the dataset), we use the number of optimizer calls, because these calls are the truly time-consuming part. This aims at comparing learning settings which involve different amounts of computation per gradient step. For instance, using SPO+, we need 2 optimizer calls to compute the loss gradient for one sample. On the other hand, we need M optimizer calls if we choose Fenchel-Young perturbed additive noise.

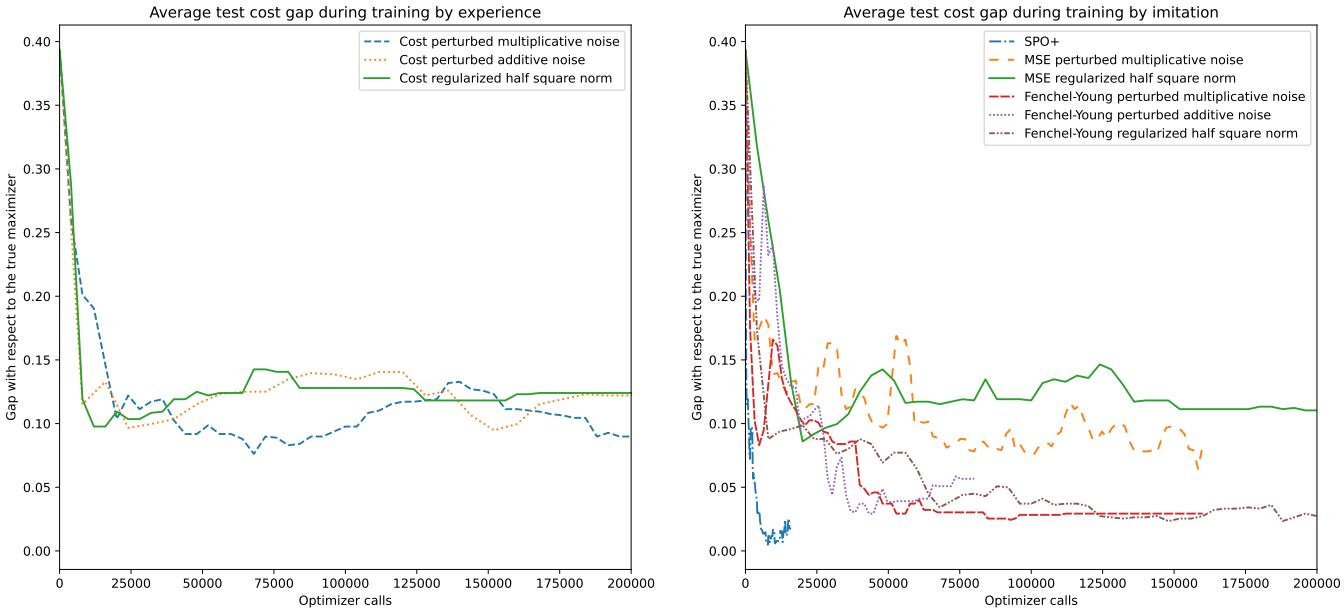
When learning by imitation, SPO+ reaches almost zero average gap both on train and test sets after very few optimizer calls, even though we only kept 1% of the initial dataset. This impressive result can be understood since, with SPO+, we have access to the true cell costs during training, and we leverage the problem structure within the loss.

Assuming we only have access to target paths, we obtain better results with Fenchel-Young losses than with MSE losses. The train and test average gaps are lower than 5% with the former, and we observe good generalization performance. This may be explained by the use of the optimization problem in the Fenchel-Young loss definition. On the contrary, in the MSE setting, although we have access to target paths, we only seek to imitate them without truly accounting for solution cost.

Perhaps surprisingly, we also manage to learn by experience with our small sub-dataset. Indeed, using the techniques introduced in Section 7.4, we reach 7% average test gaps in the cost perturbed multiplicative noise setting, which is better than learning by imitation with an MSE loss. To the best of our knowledge, it is the first time that learning by experience (as defined in Section 7.4) is combined with CNNs.



(a) Average train gap



(b) Average test gap

Figure 7.3: Train and test optimality gaps along training in the Warcraft application

8 Convex optimization with the lexicographic order

> It's very simple. Look, scissors cuts paper. Paper covers rock. Rock crushes lizard. Lizard poisons Spock. Spock smashes scissors. Scissors decapitates lizard. Lizard eats paper. Paper disproves Spock. Spock vaporizes rock. And as it always has, rock crushes scissors.
>
> *Sheldon Cooper*
> The Big Bang Theory – S2E8
> The Lizard-Spock Expansion (2008)

Contents

8.1 Introduction

Chapter 7 explained how to insert CO layers into ML pipelines, with a focus on problems that can be expressed as Linear Programs (LPs). Thanks to the regularization paradigm, we saw that convexifying a linear objective is key to obtaining meaningful derivatives.

While standard LPs cover a broad range of applications, they are sometimes insufficient for practical purposes. For instance, multiobjective LPs are very common in industrial applications, where several criteria might coexist.

8.1.1 Multiobjective optimization

The framework we focus on is a special case of vector optimization, also called multiobjective or multi-criteria optimization (Miettinen 1999; Ehrgott 2005). For lack of a natural order on the output space $\mathbb{R}^m$, vector optimization is usually concerned with finding one or several *Pareto efficient solutions*: these are solutions that cannot be strictly improved with respect to objective f_i without worsening some other objective f_j. More generally, one can be interested in minimizing f with respect to the order induced by a convex cone $\mathcal{K}$ (Bot, Grad, and Wanka 2009; Jahn 2010):

$$x \leq_{\mathcal{K}} y \quad \iff \quad y - x \in \mathcal{K}$$

We call this paradigm *conic multiobjective optimization*. Pareto efficiency corresponds to the componentwise partial order $x \leq y$, which is induced by the nonnegative orthant $\mathcal{K} = \mathbb{R}^m_+$.

8.1.2 Lexicographic optimization

In situations where the decision-maker can prioritize objectives explicitly, the *lexicographic order* $\leq_{\text{lex}}$ provides a more suitable definition of optimal solution than Pareto efficiency. For instance, when routing several trains through a railway network, one may define an importance ranking between them, specifying that the journey duration of train j matters more than that of train $j+1$. If $x_{1:j}$ denotes the subvector $(x_1, ..., x_j)$ with components 1 through j, then the lexicographic order is defined as follows:

$$x \leq_{\text{lex}} y \quad \iff \quad x = y \quad \text{or} \quad \exists j \in [m],\ x_{1:j-1} = y_{1:j-1} \text{ and } x_j < y_j.$$

Its strict counterpart excludes the equality case:

$$x <_{\text{lex}} y \quad \iff \quad \exists j \in [m],\ x_{1:j-1} = y_{1:j-1} \text{ and } x_j < y_j.$$

Here we study a *lexicographic optimization problem*: our goal is to find

$$f^\star = \operatorname*{lexmin}_{x \in \mathbb{R}^n} f(x) \quad \text{and} \quad x^\star \in \operatorname*{lexargmin}_{x \in \mathbb{R}^n} f(x),$$

where the optimal solution is selected according to the lexicographic order. If we look back to Chapter 5, we remember that the sequential approximation to MAPF can almost be thought of as a lexicographic optimization problem (5.4). This was the initial motivation for the present study.

8.1.3 Outline

In this chapter, we seek to generalize the theory of convex optimization (Boyd and Vandenberghe 2004) to vector-valued functions $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$ by endowing the output space with the lexicographic order. A crucial ingredient is the class of *lexicographically convex functions*, which is much wider than that of componentwise convex functions. We prove several useful results on this family of functions, and we design an optimization algorithm for lexicographic convex problems that does not require solving constrained convex problems. We also introduce lexicographic Fenchel-Young losses as a principled way to insert lexicographic LPs within ML pipelines.

In convex analysis, it is standard to work with extended real-valued functions. That way, constrained and unconstrained optimization share the same formalism. Similarly, we choose to work with extended vector-valued functions $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ is the completed real line. Note that the lexicographic order naturally extends to $\overline{\mathbb{R}}^m$.

8.2 Related work

The reader interested in recent overviews of multiobjective optimization can refer to the surveys by Wiecek, Ehrgott, and Engau (2016) or Eichfelder (2021).

8.2.1 Algorithms for multiobjective optimization

Since the seminal work of Geoffrion (1968), many approaches to vector optimization rely on scalarization, for instance using the weighted sum method. By combining all objectives into a single quantity, it permits the use of any scalar optimization algorithm with basically no modification.

Under additional convexity assumptions, it even recovers the entire Pareto front. However, the weighted sum method puts a lot of emphasis on the careful choice of objective weights, which can sometimes seem arbitrary.

Therefore, many authors seek to generalize standard algorithms to the multiobjective setting without resorting to scalarization. Multiobjective linear and combinatorial optimization problems in particular attract a lot of attention (Ehrgott 2005; Luc 2016). In the smooth case, numerous local search paradigms now have multiobjective counterparts: steepest descent (Fliege and Svaiter 2000; L. M. G. Drummond and Svaiter 2005), projected gradient (L. G. Drummond and A. Iusem 2004), proximal algorithms (Bonnel, A. N. Iusem, and Svaiter 2005), Newton's method (Fliege, L. M. G. Drummond, and Svaiter 2009) and so on. In the non-smooth case, subgradient algorithms (Da Cruz Neto et al. 2013; Gebken and Peitz 2021) and bundle methods also possess natural extensions.

8.2.2 Algorithms for lexicographic optimization

Given that the lexicographic order is a total order, all solutions are comparable with one another. Thus, any lexicographically optimal solution is also Pareto efficient. However, the converse is not true, which means the algorithms from Section 8.2.1, when applied with $\mathcal{K} = \mathbb{R}^m$, will not necessarily find a lexicographically optimal solution. Meanwhile, most algorithms from conic multiobjective optimization (Bot, Grad, and Wanka 2009; Jahn 2010) are designed to output a diverse set of Pareto-efficient solutions, because generic convex cones $\mathcal{K}$ do not give rise to a total order. In our case, this is overkill, because one solution is all we are looking for. Besides, several of the results from this theory expect a *closed* convex cone, which the lexicographic cone is not.

We thus need algorithms that are tailored specifically to the lexicographic setting. A natural idea would be lexicographic scalarization (Sherali and Soyster 1983; Zarepisheh and Khorram 2011), which involves putting a lot more weight on the first objective than on the second, and so on. But when there are many objectives, it often leads to numerical issues because the weights can span several orders of magnitude. Instead, the standard algorithm for lexicographic optimization is sequential (Ehrgott 2005, Algorithm 5.1):

Algorithm 8.2.1 (Sequential lexicographic optimization)**.** *For each j going from 1 to m, solve the constrained optimization problem*

$$f_j^\star = \min_{x \in \mathbb{R}^n} f_j(x) \quad s.t. \quad \forall i < j,\ f_i(x) \leq f_i^\star. \tag{8.1}$$

While the constraint $f_i(x) \leq f_i^\star$ is easy to enforce in the linear case (where $f_i(x) = c_i^\top x$), it can make optimization harder as soon as non-linearities appear. We hope to circumvent this difficulty by considering all the objectives *simultaneously* instead of sequentially.

For linear optimization, Isermann (1982) paves the way by showing that the simplex algorithm can solve lexicographic LPs directly in the lexicographic space. Tellache, Meunier, and Parmentier (2022) extend his results to tackle challenging MILPs. For convex optimization, some authors exhibit lexicographic versions of the Karush-Kuhn-Tucker conditions or duality theory (Ben-Tal and Zlobec 1977; Ben-Tal 1980; Martinez-Legaz 1988; Rentmeesters, Tsai, and K.-J. Lin 1996). Yet, to the best of our knowledge, most solution methods still rely on the sequential Algorithm 8.2.1. Furthermore, nearly all previous works make the assumption that f has convex components f_j, which is (as we will discover) a restrictive assumption. In fact, the natural class of functions to which we want to apply lexicographic minimization even contains discontinuous functions.

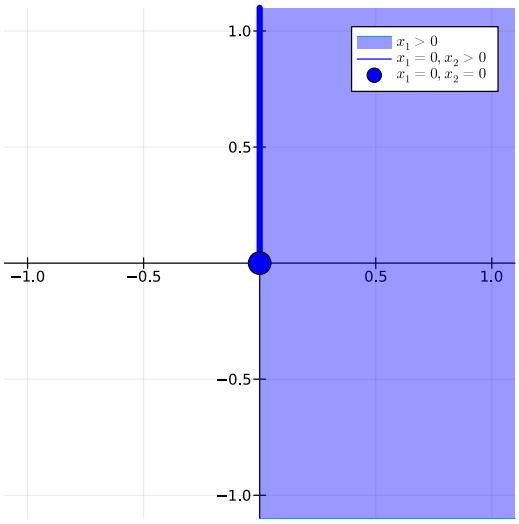


Figure 8.1: Lexicographic cone in dimension 2

8.3 Lexicographic order

Let us start by recalling some facts about our main object of interest.

8.3.1 Definitions and basic properties

Proposition 8.3.1 (Order induced by a convex cone). *Let $\mathcal{K} \subset \mathbb{R}^m$ be a pointed convex cone. Then $\mathcal{K}$ induces a partial order $\leq_{\mathcal{K}}$ on $\mathbb{R}^m$:*

$$x \leq_{\mathcal{K}} y \quad \iff \quad y - x \in \mathcal{K}.$$

This partial order is compatible with the vector space operations of $\mathbb{R}^m$.

Proof. See Bot, Grad, and Wanka (2009, Section 2.1.1). □

Definition 8.3.2 (Lexicographic cone). *The lexicographic cone $\mathcal{K}_{\text{lex}} \subset \mathbb{R}^m$ is defined by*

$$\mathcal{K}_{\text{lex}} = \{0\} \cup \{x \in \mathbb{R}^m : \exists j \in [m],\ x_{1:j-1} = 0 \text{ and } x_j > 0\}.$$

It is not hard to verify that $\mathcal{K}_{\text{lex}}$ is pointed and convex, but not closed. Indeed, $(\varepsilon, -1) \in \mathcal{K}_{\text{lex}}$ for all $\varepsilon > 0$, but $(0, -1) \notin \mathcal{K}_{\text{lex}}$. This can be seen on Figure 8.1.

Definition 8.3.3 (Lexicographic order). *The* lexicographic order *relation $\leq_{\text{lex}}$ is the order induced by the lexicographic cone $\mathcal{K}_{\text{lex}}$ on $\mathbb{R}^m$.*

The extension of $\leq_{\text{lex}}$ to $\overline{\mathbb{R}}^m$ is natural. Since $\mathcal{K}_{\text{lex}}$ is not closed, the lexicographic order is not preserved when taking limits, which will be the source of numerous problems.

Proposition 8.3.4 (Properties of the lexicographic order). *Let $x, y \in \overline{\mathbb{R}}^m$.*

1. *If $x, y \geq_{\text{lex}} 0$ and $\alpha, \beta \geq 0$, then $\alpha x + \beta y \geq_{\text{lex}} 0$.*
2. *If $x \geq_{\text{lex}} 0$, then for any $j \in [m]$, we also have $x_{1:j} \geq_{\text{lex}} 0$.*
3. *If $x \geq 0$ then $x \geq_{\text{lex}} 0$.*

Proof. Easily deduced from the definition of $\mathcal{K}_{\text{lex}}$. □

8.3.2 Link with orthogonalization

Let us recall a well-known orthogonalization algorithm.

Algorithm 8.3.5 (Gram-Schmidt orthogonalization)**.** *Given a matrix* $A = \begin{pmatrix} A_1 & \cdots & A_m \end{pmatrix} \in \mathbb{R}^{n\times m}$*, output the matrix* $Q = \begin{pmatrix} Q_1 & \cdots & Q_m \end{pmatrix} \in \mathbb{R}^{n\times m}$ *defined recursively by*

$$\forall j \in [m], \quad Q_j = A_j - \sum_{i=1}^{j-1} \mathrm{proj}_{Q_i}(A_j).$$

where proj_{Q_i} *denotes the orthogonal projection on the span of the column* Q_i*.*

We present an interesting property of the lexicographic order with respect to orthogonalization.

Proposition 8.3.6 (Stability of the lexicographic order by orthogonalization)**.** *Let* $A \in \mathbb{R}^{n\times m}$ *and* $Q = \texttt{GramSchmidt}(A)$*. For any* $h \in \mathbb{R}^n$*, if* $A^\top h \geq_{\mathrm{lex}} 0$*, then* $Q^\top h \geq_{\mathrm{lex}} 0$*.*

Proof. Let $j \in [m]$ and suppose that $Q_{1:j-1}^\top h = 0$, we will show that $Q_j^\top h \geq 0$. Since $Q_{1:j-1}^\top h = 0$, we have $h \perp \mathrm{span}(Q_1, ..., Q_{j-1})$. The Gram-Schmidt algorithm preserves the span of each subfamily, so that $\mathrm{span}(Q_1, ..., Q_{j-1}) = \mathrm{span}(A_1, ..., A_{j-1})$. Therefore, $h \perp \mathrm{span}(A_1, ..., A_{j-1})$ and $A_{1:j-1}^\top h = 0$. Our hypothesis $A^\top h \geq_{\mathrm{lex}} 0$ thus implies $A_j^\top h \geq 0$. The Gram-Schmidt algorithm ensures that $A_j = \alpha Q_j + R$ where $\alpha \geq 0$ and $R \in \mathrm{span}(A_1, ..., A_{j-1})$. We can conclude $A_j^\top h = \alpha Q_j^\top h + R^\top h = \alpha Q_j^\top h \geq 0$. □

8.3.3 Lexicographic lower bounds

Because the lexicographic order is a complete order (any two vectors are comparable), the following definitions are natural.

Definition 8.3.7 (Lex-lower bound, lex-minimum, lex-infimum)**.** *Let* $\mathcal{X} \subseteq \overline{\mathbb{R}}^m$ *and* $v \in \overline{\mathbb{R}}^m$*.*

- *If* v *satisfies* $v \leq_{\mathrm{lex}} x$ *for all* $x \in \mathcal{X}$*, we say that* v *is a* lex-lower bound *of* $\mathcal{X}$*.*
- *If* v *is a lex-lower bound of* $\mathcal{X}$ *and* v *belongs to* $\mathcal{X}$*, we say that* v *is the* lex-minimum *of* $\mathcal{X}$*, and we write* $v = \mathrm{lexmin}(\mathcal{X})$*.*
- *If* v *is the lex-maximum of all lex-lower bounds of* $\mathcal{X}$*, we say that* v *is the* lex-infimum *of* $\mathcal{X}$*, and we write* $v = \mathrm{lexinf}(\mathcal{X})$*.*

Similar definitions apply for the concepts of lex-upper bound*,* lex-maximum *and* lex-supremum*.*

If either the lex-minimum or the lex-infimum exists, it is unique. But even for bounded sets, lex-extrema can be infinite.

Example 8.3.8 (Infinite lex-extremum)**.** *Let* $\mathcal{X} = \{x \in \mathbb{R}^2 \colon 0 < x_1 \leq 1, x_2 = 0\}$*. This set is bounded in the usual sense. It is also lex-lower-bounded: the set of its lex-lower bounds is* $\{v \in \mathbb{R}^2 : v_1 \leq 0\}$*. Among these lex-lower bounds, the lex-maximum is* $(0, +\infty) = \mathrm{lexinf}(\mathcal{X})$*.*

In addition to boundedness, we can leverage an assumption of closedness to ensure the existence of a finite lex-infimum (which then becomes a lex-minimum).

Proposition 8.3.9 (Finite lex-minimum with compactness)**.** *Let $\mathcal{C} \subseteq \mathbb{R}^m$ be a non-empty compact set. Then $\mathcal{C}$ admits a finite lex-minimum.*

Proof. Let $\pi_j : x \in \mathbb{R}^m \longmapsto x_j \in \mathbb{R}$ denote the orthogonal projection on component j. We define a sequence $\mathcal{C}_j$ of subsets of $\mathcal{C}$ as follows:

$$\mathcal{C}_0 = \mathcal{C} \quad \text{and} \quad \forall j \in [m], \quad \mathcal{C}_j = \{x \in \mathcal{C}_{j-1} \colon \pi_j(x) = \min \pi_j(\mathcal{C}_{j-1})\}.$$

Suppose that $\mathcal{C}_{j-1}$ is non-empty and compact. Then, by continuity of π_j, the minimum of π_j on $\mathcal{C}_{j-1}$ (in the usual sense) exists: say it is reached at $y \in \mathcal{C}_{j-1}$. From this, we deduce that the set $\mathcal{C}_j$ is itself non-empty (because it contains y), closed (as the pre-image of $\{\min \pi_j(\mathcal{C}_{j-1})\}$ by the continuous function π_j) and bounded (as a subset of $\mathcal{C}_{j-1}$). We deduce that $\mathcal{C}_j$ is non-empty and compact as well. By recursion, this holds for all j, especially for $j = m$.

Let us conclude by proving that $\mathcal{C}_m$ is exactly the set of lex-minima of $\mathcal{C}$. To see that, consider $x \in \mathcal{C}_m$ and any other $y \in \mathcal{C}$. If $y \in \mathcal{C}_{j-1} \backslash \mathcal{C}_j$ for some $j \in [m]$, then since $\mathcal{C}_m \subseteq \mathcal{C}_j$, we have $x_{1:j-1} = y_{1:j-1}$ and $x_j < y_j$. In other words, $x <_{\text{lex}} y$. Else, $y \in \mathcal{C}_m$, which means that $x_{1:m} = y_{1:m}$, *i.e.*, $x = y$. Therefore, $\mathcal{C}$ has at least one lex-minimum, and it can have only one. This lex-minimum is finite. □

8.4 Lexicographic convexity

We now describe a natural concept of convexity with respect to the lexicographic order. It is a special case of convexity with respect to the partial order defined by a cone $\mathcal{K}$ (Bot, Grad, and Wanka 2009), but we think the case of $\mathcal{K}_{\text{lex}}$ is so unique that it deserves its own study.

Definition 8.4.1 (Lex-convexity)**.** *A function $f \colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is said to be* lex-convex *if, for all $x, y \in \mathbb{R}^n$ and all $\lambda \in [0, 1]$,*

$$f(\lambda x + (1 - \lambda)y) \leq_{\text{lex}} \lambda f(x) + (1 - \lambda) f(y).$$

If the inequality is strict for all $x \neq y$ whenever $\lambda \in (0, 1)$, we say that f is strictly lex-convex.

Definition 8.4.2 (Domain, properness)**.** *The* domain *of a function $f \colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is the subset of $\mathbb{R}^n$ where all of its components take finite values:*

$$\operatorname{dom}(f) = \{x \in \mathbb{R}^n : \forall j \in [m], \ -\infty < f_j(x) < +\infty\}$$

Furthermore, f is said to be proper *if none of its components ever takes the value $-\infty$, and if its domain is non-empty.*

In the following, every lex-convex function we consider is assumed to be proper unless otherwise specified.

8.4.1 Lexicographic epigraph

As in the scalar case, there is a strong connection between lex-convex functions and convex sets.

Definition 8.4.3 (Lex-epigraph)**.** *The* lex-epigraph *of a function $f \colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is defined by*

$$\operatorname{lexepi}(f) = \{(x, v) \in \mathbb{R}^n \times \overline{\mathbb{R}}^m : f(x) \leq_{\text{lex}} v\} \subseteq \overline{\mathbb{R}}^m.$$

Proposition 8.4.4 (Lex-convexity and lex-epigraph)**.** *A function $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is lex-convex if and only if its lex-epigraph is a convex set.*

Proof. First, suppose that f is lex-convex. Let (x, u) and (y, v) be two elements of $\mathrm{lexepi}(f)$, and $\lambda \in [0, 1]$. By lex-convexity of f, we know that

$$f(\lambda x + (1-\lambda)y) \leq_{\mathrm{lex}} \lambda f(x) + (1-\lambda) f(y).$$

And because $(x, u) \in \mathrm{lexepi}(f)$ (resp. $(y, v) \in \mathrm{lexepi}(f)$), we have $f(x) \leq_{\mathrm{lex}} u$ (resp. $f(y) \leq_{\mathrm{lex}} v$). Therefore,

$$f(\lambda x + (1-\lambda)y) \leq_{\mathrm{lex}} \lambda u + (1-\lambda) v,$$

which amounts to

$$\begin{pmatrix} \lambda x + (1-\lambda)y \\ \lambda u + (1-\lambda)v \end{pmatrix} \in \mathrm{lexepi}(f).$$

In other words, $\mathrm{lexepi}(f)$ is a convex set.

Conversely, suppose that $\mathrm{lexepi}(f)$ is a convex set. Let $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$. We define $u = f(x)$ and $v = f(y)$, so that (x, u) and (y, v) belong to $\mathrm{lexepi}(f)$. By convexity of $\mathrm{lexepi}(f)$, we get

$$\begin{pmatrix} \lambda x + (1-\lambda)y \\ \lambda u + (1-\lambda)v \end{pmatrix} \in \mathrm{lexepi}(f).$$

We deduce

$$f(\lambda x + (1-\lambda)y) \leq \lambda u + (1-\lambda) v = \lambda f(x) + (1-\lambda) f(y).$$

In other words, f is lex-convex. □

8.4.2 Examples and composition rules

Here we outline several ways to build lex-convex functions, as well as some caveats associated with this very wide class.

Proposition 8.4.5 (Lex-convex subfunction)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a lex-convex function. Then for any $j \in [m]$, the subfunction $f_{1:j}\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^j$ is also lex-convex.*

Proof. Direct consequence of Proposition 8.3.4. □

Proposition 8.4.6 (Componentwise convex function)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a function whose components $f_j : \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}$ are all convex. Then f is lex-convex.*

Proof. The following inequality holds componentwise:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

By Proposition 8.3.4, it also holds in a lexicographic sense. This means f is lex-convex. □

Proposition 8.4.7 (Affine function)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be an affine function. Then f is lex-convex.*

Proof. Direct consequence of Proposition 8.4.6. □

Proposition 8.4.8 (Affine composition)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a lex-convex function and $\psi\colon x \in \mathbb{R}^p \longmapsto Ax + b \in \mathbb{R}^n$ be an affine function. Then $f \circ \psi\colon \mathbb{R}^p \longrightarrow \overline{\mathbb{R}}^m$ is lex-convex.*

Proof. By lex-convexity of f,

$$\begin{aligned}(f \circ \psi)(\lambda x + (1-\lambda)y) &= f\left(A(\lambda x + (1-\lambda)y) + b\right)\\ &= f\left(\lambda(Ax+b) + (1-\lambda)(Ay+b)\right)\\ &\leq_{\text{lex}} \lambda f(Ax+b) + (1-\lambda) f(Ay+b)\\ &= \lambda(f\circ\psi)(x) + (1-\lambda)(f\circ\psi)(y).\end{aligned}$$

This means $f \circ \psi$ is lex-convex. □

Proposition 8.4.9 (Convex function times vector)**.** *Let $\psi\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}$ be a real-valued function, and $v \in \mathbb{R}^m$ be a vector. If ψ is convex and $v \geq_{\text{lex}} 0$, then $f\colon x \in \mathbb{R}^n \longmapsto \psi(x)v \in \overline{\mathbb{R}}^m$ is lex-convex.*

Proof. By convexity of ψ, for all $x, y \in \mathbb{R}^n$ and all $\lambda \in [0,1]$,

$$a = [\lambda\psi(x) + (1-\lambda)\psi(y)] - \psi(\lambda x + (1-\lambda)y) \geq 0.$$

Since $v \geq_{\text{lex}} 0$, we also have $av \geq_{\text{lex}} 0$, which shows that

$$[\lambda f(x) + (1-\lambda) f(y)] - f(\lambda x + (1-\lambda)y) \geq_{\text{lex}} 0.$$

Therefore, f is lex-convex. □

Proposition 8.4.10 (Sum of lex-convex functions)**.** *Let f^1, f^2 be two lex-convex functions from $\mathbb{R}^n$ to $\overline{\mathbb{R}}^m$, and let $\alpha, \beta \geq 0$ be two real numbers. Then the weighted sum $\alpha f^1 + \beta f^2$ is also lex-convex.*

Proof. By Proposition 8.3.1, the lexicographic order is compatible with the vector space operations of $\mathbb{R}^m$. Nonnegative scaling and addition do not give rise to ambiguous quantities of the form $\infty - \infty$, so this result extends to $\overline{\mathbb{R}}^m$. □

Proposition 8.4.11 (Lexmax of lex-convex functions)**.** *Let $(f^k)_{k\in\mathcal{K}}$ be a (possibly uncountable) family of lex-convex functions from $\mathbb{R}^n$ to $\overline{\mathbb{R}}^m$. Then their pointwise lex-maximum $f\colon x \in \mathbb{R}^n \longmapsto \operatorname{lexmax}_{k\in\mathcal{K}} f^k(x) \in \overline{\mathbb{R}}^m$ is also lex-convex.*

Proof. By definition of the lex-epigraph,

$$\begin{aligned}\operatorname{lexepi}(f) &= \{(x,v) \in \mathbb{R}^n \times \overline{\mathbb{R}}^m : \operatorname*{lexmax}_{k\in\mathcal{K}} f^k(x) \leq_{\text{lex}} v\}\\ &= \{(x,v) \in \mathbb{R}^n \times \overline{\mathbb{R}}^m : \forall k \in \mathcal{K},\ f^k(x) \leq_{\text{lex}} v\}\\ &= \bigcap_{k\in\mathcal{K}} \operatorname{lexepi}(f^k)\end{aligned}$$

By Proposition 8.4.4, since each f^k is lex-convex, its lex-epigraph is a convex set. Therefore, the intersection of these epigraphs is a convex set as well. Using Proposition 8.4.4 again in the opposite direction, we conclude that f is lex-convex. □

8.4.3 Coarser than componentwise convexity

We proceed to show that the family of lex-convex functions includes componentwise convex functions, but not only. We give two counterexamples.

Example 8.4.12 (Lex-convex function which is not componentwise convex)**.** *The function* $f\colon x \in \mathbb{R} \longmapsto (x^2, -x^2)$ *is lex-convex. To see it, apply Proposition 8.4.9 with* $v = (1,-1) >_{\text{lex}} 0$*. However, its second component is strictly concave and hence not convex.*

Example 8.4.13 (Lex-convex function which is not continuous)**.** *Consider the two affine functions*

$$f^1\colon (x_1,x_2) \longmapsto (x_1,0) \quad \textit{and} \quad f^2\colon (x_1,x_2) \longmapsto (0,x_2).$$

By Proposition 8.4.7, they are both lex-convex, and by Proposition 8.4.11 their pointwise lex-maximum $f = \operatorname{lexmax}\{f^1, f^2\}$ *is lex-convex as well. However,* h *is not continuous:*

$$f(\varepsilon,1) = (\varepsilon,0) \xrightarrow[\varepsilon \longrightarrow 0]{} (0,0) \neq (0,1) = f(0,1)$$

The following result can be seen as a partial converse to Proposition 8.4.6.

Proposition 8.4.14 (Restricted convexity of the last component)**.** *Let* $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ *be a lex-convex function and* $j \in [m]$*. Suppose that* $f_{1:j-1}$ *is lex-lower-bounded by* $v \in \mathbb{R}^{j-1}$ *on* $\mathbb{R}^n$*: for all* $x \in \mathbb{R}^n$*, we have* $f_{1:j-1}(x) \geq_{\text{lex}} v$*. We define* $\mathcal{X}_j$ *as the set of inputs where this lex-lower bound is tight:* $\mathcal{X}_j = \{x \in \mathbb{R}^n : f_{1:j-1}(x) = v\}$*. Then* $\mathcal{X}_j$ *is a convex set and the component* f_j *is convex on* $\mathcal{X}_j$

Proof. Let x and y be two elements of $\mathcal{X}_j$, and $\lambda \in [0,1]$. By Proposition 8.4.5, since f is lex-convex, $f_{1:j}$ and $f_{1:j-1}$ are also lex-convex. By lex-convexity of $f_{1:j-1}$,

$$v \leq f_{1:j-1}(\lambda x + (1-\lambda)y) \leq_{\text{lex}} \lambda f_{1:j-1}(x) + (1-\lambda) f_{1:j-1}(y) = \lambda v + (1-\lambda)v = v.$$

As a result, $\lambda x + (1-\lambda)y \in \mathcal{X}_j$, and so $\mathcal{X}_j$ is a convex set. Furthermore, by lex-convexity of $f_{1:j}$,

$$\begin{pmatrix} f_{1:j-1}(\lambda x + (1-\lambda)y) \\ f_j(\lambda x + (1-\lambda)y) \end{pmatrix} \leq_{\text{lex}} \lambda \begin{pmatrix} f_{1:j-1}(x) \\ f_j(x) \end{pmatrix} + (1-\lambda) \begin{pmatrix} f_{1:j-1}(y) \\ f_j(y) \end{pmatrix}.$$

But since x, y and $\lambda x + (1-\lambda)y$ all belong to $\mathcal{X}_j$, the value of $f_{1:j-1}$ at these three points is the same:

$$\begin{pmatrix} v \\ f_j(\lambda x + (1-\lambda)y) \end{pmatrix} \leq_{\text{lex}} \lambda \begin{pmatrix} v \\ f_j(x) \end{pmatrix} + (1-\lambda) \begin{pmatrix} v \\ f_j(y) \end{pmatrix}.$$

By definition of the lexicographic order, we have the following constraint on f_j:

$$f_j(\lambda x + (1-\lambda)y) \leq \lambda f_j(x) + (1-\lambda) f_j(y).$$

In other words, f_j is convex on $\mathcal{X}_j$. □

This has important consequences for the sequential Algorithm 8.2.1.

Theorem 8.4.15 (Sequential minimization works for lex-convex functions)**.** *When applied to a lex-convex function, the sequential Algorithm 8.2.1 solves a sequence of constrained convex minimization problems.*

Proof. Directly implied by Proposition 8.4.14. □

8.5 Lexicographic minimization

As we will see later on, lex-convex functions can appear naturally in some optimization problems. It thus makes sense to ask whether they always have a (unique) minimizer.

8.5.1 Lexicographic coercivity and strong convexity

The usual notion of coercivity has a natural counterpart in the lexicographic setting.

Definition 8.5.1 (Lex-coercivity)**.** *A function $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is said to be* lex-coercive *if for all $v \in \mathbb{R}^m$, there is a radius $r \in [0, +\infty)$ such that $\|x\| \geq r$ implies $f(x) \geq_{\text{lex}} v$.*

Proposition 8.5.2 (Existence of a lex-minimizer for lex-coercive functions)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper, continuous and lex-coercive function. Then the lexicographic optimization problem $\operatorname{lexmin}_{x\in\mathbb{R}^n} f(x)$ has at least one optimal solution.*

Proof. Since f is proper, there is an $x^0 \in \mathbb{R}^n$ such that $f(x^0)$ is finite. By lex-coercivity, there is a radius $r \in [0, \infty)$ such that $\|x\| \geq r$ implies $f(x) \geq_{\text{lex}} f(x^0)$. We define $\mathcal{X} = \mathcal{B}(x^0, r)$, which is a non-empty compact set, and $\mathcal{C} = f(\mathcal{X})$. By continuity of f, the set $\mathcal{C}$ is also non-empty and compact. By Proposition 8.3.9, $\mathcal{C}$ has a lex-minimum v. Since $\mathcal{C} = f(\mathcal{X})$, there is an $x \in \mathcal{X}$ such that $f(x) = v$. For all $y \in \mathcal{X}$, we have $f(y) \geq_{\text{lex}} v = f(x)$. And for all $y \in \mathbb{R}^n \backslash \mathcal{X}$, we have $f(y) \geq_{\text{lex}} f(x^0) \geq_{\text{lex}} v = f(x)$ (remember that $x^0 \in \mathcal{X}$). Hence, x is an optimal solution to the lexicographic optimization problem $\operatorname{lexmin}_{x\in\mathbb{R}^n} f(x)$. □

To generalize γ-strong convexity, we need to replace the curvature parameter $\gamma > 0$ with a curvature vector $v >_{\text{lex}} 0$.

Definition 8.5.3 (Strong lex-convexity)**.** *Let $v \in \overline{\mathbb{R}}^m$ be such that $v >_{\text{lex}} 0$. A function $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ is said to be* v-strongly lex-convex *if for all $x, y \in \mathbb{R}^n$, for all $\lambda \in [0, 1]$,*

$$f(\lambda x + (1-\lambda)y) \leq_{\text{lex}} \lambda f(x) + (1-\lambda) f(y) - \frac{\lambda(1-\lambda)}{2}\|x-y\|^2 v.$$

Proposition 8.5.4 (Strong lex-convexity implies strict lex-convexity)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be v-strongly lex-convex for some $v >_{\text{lex}} 0$. Then f is also strictly lex-convex.*

Proof. Whenever $\lambda \in (0, 1)$ and $x \neq y$, since $v >_{\text{lex}} 0$ we have $\frac{\lambda(1-\lambda)}{2}\|x-y\|^2 v >_{\text{lex}} 0$. Plugging this into the inequality defining v-strong lex-convexity implies

$$f(\lambda x + (1-\lambda)y) <_{\text{lex}} \lambda f(x) + (1-\lambda) f(y).$$

□

Proposition 8.5.5 (Strong lex-convexity implies lex-coercivity)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be proper, continuous and v-strongly lex-convex for some $v >_{\text{lex}} 0$. Then f is also lex-coercive.*

Proof. Since f is proper and continuous, there is an $x^0 \in \mathbb{R}^n$ such that $f(x^0)$ is finite. By continuity, there is a radius $r \in (0, +\infty)$ such that f remains finite on the ball $\mathcal{B}(x^0, r)$. Without loss of

generality, we can take $x^0 = 0$ and $r = 1$ to simplify. Given $x \in \mathbb{R}^n$, we write the v-strong lex-convexity condition with $\lambda = 1/\|x\|$ and $y = 0$:

$$f\left(\frac{1}{\|x\|}x\right) \leq_{\text{lex}} \frac{1}{\|x\|}f(x) + \left(1 - \frac{1}{\|x\|}\right)f(0) - \frac{\frac{1}{\|x\|}\left(1 - \frac{1}{\|x\|}\right)}{2}\|x\|^2 v.$$

Since f is continuous and finite on $\mathcal{B}(0,1)$, we know that $|f|(\mathcal{B}(0,1))$ is also non-empty and compact. By Proposition 8.3.9, this image ball admits a lex-maximum z. We can therefore write:

$$\begin{aligned} f(x) &\geq_{\text{lex}} \|x\| f\left(\frac{1}{\|x\|}x\right) - (\|x\| - 1)f(0) + \frac{1}{2}\left(1 - \frac{1}{\|x\|}\right)\|x\|^2 v \\ &\geq_{\text{lex}} -2\|x\|z + \frac{1}{4}\|x\|^2 v. \end{aligned}$$

Since $v >_{\text{lex}} 0$, the function $x \longmapsto -2\|x\|z + \frac{1}{4}\|x\|^2 v$ is lex-coercive, and so is f. □

Proposition 8.5.6 (Convex function times vector – strong lex-convexity)**.** *Consider the setting of Proposition 8.4.9. If ψ is γ-strongly convex and $v >_{\text{lex}} 0$, then $f(x) = \psi(x)v$ is γv-strongly lex-convex.*

Proof. Since ψ is γ-strongly convex,

$$a = \left[\lambda\psi(x) + (1-\lambda)\psi(y) - \frac{\lambda(1-\lambda)}{2}\|x - y\|^2\gamma\right] - \psi(\lambda x + (1-\lambda)y) \geq 0.$$

Using the fact that $v \geq_{\text{lex}} 0$, we obtain $av \geq_{\text{lex}} 0$, which shows that f is γv-strongly lex-convex. □

8.5.2 Existence and uniqueness of a minimizer

We now have all the tools we need to ensure that lexicographic optimization problems are well-defined.

Proposition 8.5.7 (Strict lex-convexity implies minimizer uniqueness)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be strictly lex-convex. If the optimal solution to the problem $\operatorname{lexmin}_{x\in\mathbb{R}^n} f(x)$ exists, then it is unique.*

Proof. Suppose x and y are distinct lex-minimizers of f on $\mathbb{R}^n$. By strict lex-convexity, we would have

$$f\left(\frac{x+y}{2}\right) <_{\text{lex}} \frac{f(x) + f(y)}{2},$$

which is impossible. □

Theorem 8.5.8 (Minimization of lex-convex functions)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper, continuous, lex-coercive and strictly lex-convex function. Then the problem $\operatorname{lexmin}_{x\in\mathbb{R}^n} f(x)$ has an optimal solution, and this solution is unique.*

Proof. By Proposition 8.5.2, the minimizer exists. By Proposition 8.5.7, the minimizer is unique. □

Proposition 8.5.9 (Minimization of strongly lex-convex functions)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper, continuous and strongly lex-convex function. Then the problem $\operatorname{lexmin}_{x\in\mathbb{R}^n} f(x)$ has an optimal solution, and this solution is unique.*

Proof. Combine Propositions 8.5.4 and 8.5.5 with Theorem 8.5.8. □

8.5.3 Approximate minimization

In linear optimization, the simplex algorithm returns a combinatorial object called a basis, which encodes the optimal solution in an exact manner. On the other hand, convex optimization is approximate: there will always be some numerical error keeping us from reaching the exact optimum. This means that when running the sequential Algorithm 8.2.1, the constraint $f_i(x) = f_i^\star$ cannot be enforced with infinite precision. As a result, we need a notion of approximate optimality to analyze the convergence of our algorithms.

Definition 8.5.10 (Approximate lex-minimizer)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a function with lex-minimum $f^\star$. A vector $x \in \mathbb{R}^n$ is an* ε-approximate lex-minimizer *if it satisfies $f(x) \leq f^\star + \varepsilon\mathbf{1}$.*

We stress that the inequality in the previous definition must hold componentwise.

8.6 Lexicographic subgradients

Now that we have discussed the existence of a lex-minimizer, we ask how to compute it. Let us start by generalizing the standard notion of subgradient.

Definition 8.6.1 (Lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper function. A matrix $G \in \mathbb{R}^{n\times m}$ is a* lex-subgradient *of f at $x \in \operatorname{dom}(f)$ if, for all $y \in \mathbb{R}^n$,*

$$f(y) - f(x) \geq_{\text{lex}} G^\top (y - x).$$

We denote by $\partial^{\text{lex}} f(x)$ the lex-subdifferential *of f at x, which is the set of its lex-subgradients.*

Note the major difference with the real-valued case: while usual subgradients are just *vectors*, lex-subgradients are *matrices*. The lex-subdifferential is always a convex set, but it may not be closed nor bounded (see Example 8.6.13). And for optimization purposes, it is of particular interest to know whether it contains 0.

Proposition 8.6.2 (Unconstrained lex-optimality condition)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function. Then $x \in \operatorname{dom}(f)$ is a global lex-minimizer of f if and only if $0 \in \partial^{\text{lex}} f(x)$.*

Proof. We have the following equivalence relations:

$$\begin{aligned} x \text{ is a global lex-minimizer of } f \quad &\iff \quad \forall y \in \mathbb{R}^n,\ f(y) \geq_{\text{lex}} f(x) \\ &\iff \quad \forall y \in \mathbb{R}^n,\ f(y) - f(x) \geq_{\text{lex}} 0^\top (y - x) \\ &\iff \quad 0 \in \partial^{\text{lex}} f(x) \end{aligned}$$

□

8.6.1 Characterization and existence

Proposition 8.6.3 (Columns of a lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function. Let $x \in \operatorname{dom}(f)$ and $G = \begin{pmatrix} G_1 & \cdots & G_m \end{pmatrix} \in \mathbb{R}^{n\times m}$ For all $j \in [m]$, we define*

$$\mathcal{Y}_j = \{y \in \mathbb{R}^n : f_{1:j-1}(y) - f_{1:j-1}(x) = G_{1:j-1}^\top (y - x)\}.$$

Then the following two statements are equivalent:

1. *The matrix G is a lex-subgradient of f at x*
2. *For all $j \in [m]$,*
   (a) *the set $\mathcal{Y}_j$ is convex*
   (b) *the function f_j is convex on $\mathcal{Y}_j$*
   (c) *the column G_j is a subgradient of $f_{j|\mathcal{Y}_j}$ at x.*

Proof. We start by noting that for all $j \in [m]$, the vector x belongs to $\mathcal{Y}_j$.

$1 \implies 2$**:** Let $G \in \partial^{\text{lex}} f(x)$: the function $y \longmapsto f(y) - G^\top y$ is lex-convex. We fix $j \in [m]$.

By Proposition 8.3.4, $G_{1:j-1} \in \partial^{\text{lex}} f_{1:j-1}(x)$. Therefore, $y \longmapsto f_{1:j-1}(y) - G_{1:j-1}^\top y$ is lex-lower-bounded by $v = f_{1:j-1}(x) - G_{1:j-1}(x)$ on $\mathbb{R}^n$. Since $\mathcal{Y}_j$ is the set where this lex-lower bound is tight, by Proposition 8.4.14, it is a convex set and $y \longmapsto f_j(y) - G_j^\top y$ is convex on $\mathcal{Y}_j$. This also implies that f_j is convex on $\mathcal{Y}_j$.

By Proposition 8.3.4, $G_{1:j} \in \partial^{\text{lex}} f_{1:j}(x)$. Therefore, for every $y \in \mathbb{R}^n$,

$$\begin{pmatrix} f_{1:j-1}(y) - f_{1:j-1}(x) \\ f_j(y) - f_j(x) \end{pmatrix} = f_{1:j}(y) - f_{1:j}(x) \geq_{\text{lex}} G_{1:j}^\top (y - x) = \begin{pmatrix} G_{1:j-1}^\top (y - x) \\ G_j^\top (y - x) \end{pmatrix},$$

which we can rewrite as

$$\begin{pmatrix} f_{1:j-1}(y) - f_{1:j-1}(x) - G_{1:j-1}^\top (y - x) \\ f_j(y) - f_j(x) - G_j^\top (y - x) \end{pmatrix} \geq_{\text{lex}} 0.$$

By definition of $\mathcal{Y}_j$, the first block is zero for all $y \in \mathcal{Y}_j$. In order for the whole vector to be $\geq_{\text{lex}} 0$, the last component must satisfy

$$f_j(y) - f_j(x) - G_j^\top (y - x) \geq 0.$$

This proves that G_j is a subgradient of $f_{j|\mathcal{Y}_j}$ at x.

$2 \implies 1$**:** We prove this implication by recursion on m. The initialization $m = 1$ is obvious. Suppose the result is true for $m - 1$, and let $G \in \mathbb{R}^{n \times m}$ be such that $\mathcal{Y}_j$ is convex, f_j is convex on $\mathcal{Y}_j$ and $G_j \in \partial f_{j|\mathcal{Y}_j}(x)$ for every $j \in [m]$.

Using our recursion hypothesis, we find that $G_{1:m-1} \in \partial^{\text{lex}} f_{1:m-1}(x)$: for all $y \in \mathbb{R}^n$,

$$f_{1:m-1}(y) - f_{1:m-1}(x) \geq_{\text{lex}} G_{1:m-1}^\top (y - x).$$

By assumption, we also know that $G_m \in \partial f_{m|\mathcal{Y}_m}(x)$: for all $y \in \mathcal{Y}_m$,

$$f_m(y) - f_m(x) \geq_{\text{lex}} G_m^\top (y - x).$$

So for a given $y \in \mathbb{R}^n$, one of two things happens:

- If $y \in \mathcal{Y}_m$, then $f_{1:m-1}(y) - f_{1:m-1}(x) = G_{1:m-1}^\top (y - x)$ and $f_m(y) - f_m(x) \geq G_m^\top (y - x)$.
- If $y \notin \mathcal{Y}_m$, then $f_{1:m-1}(y) - f_{1:m-1}(x) >_{\text{lex}} G_{1:m-1}^\top (y - x)$.

In both cases, we have

$$f(y) - f(x) = \begin{pmatrix} f_{1:m-1}(y) - f_{1:m-1}(y) \\ f_m(y) - f_m(x) \end{pmatrix} \geq_{\text{lex}} \begin{pmatrix} G_{1:m-1}^\top (y-x) \\ G_m^\top (y-x) \end{pmatrix} = G^\top (y-x).$$

This proves that the matrix G is a lex-subgradient of f at x.

□

For real-valued functions, convexity implies the existence of subgradients at every point. It is reassuring to see that this extends to the lexicographic case.

Proposition 8.6.4 (Existence of a lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function. For every $x \in \operatorname{dom}(f)$, the subdifferential $\partial^{\text{lex}} f(x)$ is non-empty.*

Proof. We prove this by recursion on m. The scalar case $m = 1$ is well-known (Bubeck 2015, Proposition 1.1). Suppose the result is true for $m-1$, and let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a lex-convex function. Thanks to Proposition 8.3.4, we know that $f_{1:m-1}$ is also lex-convex. Our recursion hypothesis implies the existence of a lex-subgradient $G_{1:m-1} \in \partial^{\text{lex}} f_{1:m-1}(x)$. For all $y \in \mathbb{R}^n$,

$$f_{1:m-1}(y) - f_{1:m-1}(x) \geq_{\text{lex}} G_{1:m-1}^\top (y-x).$$

Let us denote by $\mathcal{Y}_m$ the set of inputs where this lex-lower bound is tight:

$$\mathcal{Y}_m = \{y \in \mathbb{R}^n : f_{1:m-1}(y) - f_{1:m-1}(x) = G_{1:m-1}^\top (y-x)\}.$$

The function

$$y \longmapsto f(y) - \begin{pmatrix} G_{1:m-1} & 0 \end{pmatrix}^\top y = \begin{pmatrix} f_{1:m-1}(y) - G_{1:m-1}^\top y \\ f_m(y) - 0^\top y \end{pmatrix}$$

is lex-convex, and its first block is lex-lower-bounded by $v = f_{1:m-1}(x) - G_{1:m-1}^\top x$ on $\mathbb{R}^n$. Thus, by Proposition 8.4.14, the set $\mathcal{Y}_m$ is convex and the last component f_m is convex on $\mathcal{Y}_m$. Using the result for $m = 1$ once again, we find that the function $f_{m|\mathcal{Y}_m}$ admits a subgradient g at x: for all $y \in \mathcal{Y}_m$,

$$f_m(y) - f_m(x) \geq g^\top (y-x).$$

We now construct the matrix

$$G = \begin{pmatrix} G_{1:m-1} & g \end{pmatrix} \in \mathbb{R}^{n \times m}.$$

By Proposition 8.6.3 ($1 \implies 2$), we know that $G_j \in \partial f_{j|\mathcal{Y}_j}(x)$ for all $j \in [m-1]$, where $\mathcal{Y}_j$ is defined similarly to $\mathcal{Y}_m$. But we have just seen that $g = G_m \in \partial f_{m|\mathcal{Y}_m}(x)$ too. So we can apply Proposition 8.6.3 ($2 \implies 1$) and conclude that G is a lex-subgradient of f at x. □

The previous result outlines a procedure for finding a lex-subgradient, and by Proposition 8.6.3, every lex-subgradient can be constructed in this way. However, it is not very practical, which is why lex-subgradients are easier to compute using composition rules such as the ones given below.

8.6.2 Examples and composition rules

We now cycle through Section 8.4.2 and present ways to compute a lex-subgradient for each function given there. Note however that we do not provide the full lex-subdifferential.

Proposition 8.6.5 (Lex-convex subfunction – lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function, and let $G \in \partial^{\text{lex}} f(x)$ for some $x \in \text{dom}(f)$. Then for any $j \in [m]$, we have $G_{1:j} \in \partial^{\text{lex}} f_{1:j}(x)$.*

Proof. We know from Proposition 8.4.5 that f is convex. Proposition 8.3.4 implies that $G \in \partial^{\text{lex}} f(x)$. □

Proposition 8.6.6 (Componentwise convex function – lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a function whose components $f_j : \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}$ are all proper and convex. For some $x \in \text{dom}(f)$, let $g_j \in \partial f_j(x)$ for all $j \in [m]$. Then $G = \begin{pmatrix} g_1 & \cdots & g_m \end{pmatrix} \in \partial^{\text{lex}} f(x)$.*

Proof. We know from Proposition 8.4.6 that f is convex. The following inequality holds componentwise:

$$f(y) - f(x) \geq G^\top (y - x).$$

Proposition 8.3.4 implies that $G \in \partial^{\text{lex}} f(x)$. □

Proposition 8.6.7 (Affine function – lex-subgradient)**.** *Let $f\colon x \in \mathbb{R}^n \longmapsto Ax + b \in \mathbb{R}^m$ be an affine function. Then $A^\top \in \partial^{\text{lex}} f(x)$ for all $x \in \mathbb{R}^n$.*

Proof. We know from Proposition 8.4.7 that f is lex-convex. Proposition 8.6.6 implies that $A^\top \in \partial^{\text{lex}} f(x)$. □

Proposition 8.6.8 (Affine composition – lex-subgradient)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function and $\psi\colon x \in \mathbb{R}^p \longmapsto Ax + b \in \mathbb{R}^n$ be an affine function. For $x \in \psi^{-1}(\text{dom}(f))$, let $G \in \partial^{\text{lex}} f(\psi(x))$. Then $A^\top G \in \partial^{\text{lex}}(f \circ \psi)(x)$.*

Proof. We know from Proposition 8.4.8 that $f \circ \psi$ is lex-convex. Furthermore,

$$(f \circ \psi)(y) - (f \circ \psi)(x) = f(Ay + b) - f(Ax + b) \geq_{\text{lex}} G^\top (Ay - Ax) = (A^\top G)^\top (y - x)$$

This means that $A^\top G \in \partial^{\text{lex}}(f \circ \psi)(x)$. □

Proposition 8.6.9 (Convex function times vector – lex-subgradient)**.** *Let $\psi\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}$ be a proper convex function, and $v \in \mathbb{R}^m$ be such that $v \geq_{\text{lex}} 0$, If g is a subgradient of ψ at some $x \in \text{dom}(\psi)$, then $G = gv^\top$ is a lex-subgradient of f at x.*

Proof. We know from Proposition 8.4.9 that f is lex-convex. For all $y \in \mathbb{R}^n$,

$$\psi(y) - \psi(x) \geq g^\top (y - x).$$

As a consequence,

$$f(y) - f(x) \geq_{\text{lex}} (g^\top (y - x))v.$$

It is easily seen that

$$\Big[(g^\top (y - x))v\Big]_j = \left(\sum_i g_i (y_i - x_i)\right) v_j = \sum_i (g_i v_j)\,(y_i - x_i) = [(vg^\top)(y - x)]_j.$$

Therefore, we have

$$f(y) - f(x) \geq_{\text{lex}} G^\top (y - x) \quad \text{with} \quad G = gv^\top,$$

and $G = gv^\top \in \mathbb{R}^{n \times m}$ is a lex-subgradient of f at x. □

Proposition 8.6.10 (Sum of lex-convex functions – lex-subgradient)**.** *Let f^1, f^2 be two proper lex-convex functions from $\mathbb{R}^n$ to $\overline{\mathbb{R}}^m$, and let α, β be two nonnegative real numbers. For some $x \in \text{dom}(f^1) \cap \text{dom}(f^2)$, let $G^1 \in \partial^{\text{lex}} f^1(x)$ and $G^2 \in \partial^{\text{lex}} f^2(x)$. Then $\alpha G^1 + \beta G^2 \in \partial^{\text{lex}}(\alpha f^1 + \beta f^2)(x)$.*

Proof. We know from Proposition 8.4.10 that $\alpha f^1 + \beta f^2$ is lex-convex. For any $y \in \mathbb{R}^n$,

$$(\alpha f^1 + \beta f^2)(y) - (\alpha f^1 + \beta f^2)(x) \geq_{\text{lex}} (\alpha G^1 + \beta G^2)^\top (y - x),$$

which shows that $\alpha G^1 + \beta G^2 \in \partial^{\text{lex}}(\alpha f^1 + \beta f^2)(x)$. □

Proposition 8.6.11 (Lexmax of lex-convex functions – lex-subgradient)**.** *Let $(f^k)_{k \in \mathcal{K}}$ be a (possibly uncountable) family of proper lex-convex functions from $\mathbb{R}^n$ to $\overline{\mathbb{R}}^m$. Let $f \colon x \in \mathbb{R}^n \longmapsto \text{lexmax}_{k \in \mathcal{K}} f^k(x)$ denote their pointwise lex-maximum. For some $x \in \text{dom}(f)$, let $k \in \mathcal{K}$ be such that $f^k(x) = f(x)$. If $G^k \in \partial^{\text{lex}} f^k(x)$, then $G^k \in \partial^{\text{lex}} f(x)$.*

Proof. We know from Proposition 8.4.11 that f is lex-convex. Suppose $f^k(x) = f(x)$. For any $y \in \mathbb{R}^n$, it holds that

$$f(y) - f(x) = f(y) - f^k(x) \geq_{\text{lex}} f^k(y) - f^k(x) \geq_{\text{lex}} (G^k)^\top (y - x)$$

which proves that $G^k \in \partial^{\text{lex}} f(x)$. □

8.6.3 Link with differentiability

When f is differentiable, some of the lex-subgradients given above can be recovered using the Jacobian matrix.

Proposition 8.6.12 (The case of differentiable lex-convex functions)**.** *Let $f \colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper differentiable lex-convex function. Then for all $x \in \text{dom}(f)$, we have $\text{J}f(x)^\top \in \partial^{\text{lex}} f(x)$.*

Proof. By definition of the Jacobian matrix,

$$G = \text{J}f(x)^\top = \begin{pmatrix} \nabla f_1(x) & \cdots & \nabla f_m(x) \end{pmatrix}.$$

For all $j \in [m]$, we define

$$\mathcal{Y}_j = \{y \in \mathbb{R}^n : f_{1:j-1}(y) - f_{1:j-1}(x) = G_{1:j-1}^\top (y - x)\}.$$

By an argument similar to the proof of Proposition 8.6.4, we deduce from Proposition 8.4.14 that the set $\mathcal{Y}_j$ is convex and that f_j is convex on $\mathcal{Y}_j$. Since f_j is differentiable on $\text{dom}(f)$, its gradient $G_j = \nabla f_j(x)$ is also a subgradient of $f_{j|\mathcal{Y}_j}$ at $x \in \mathcal{Y}_j$: for all $y \in \mathcal{Y}_j$,

$$f_j(y) - f_j(x) \geq \nabla f_j(x)^\top (y - x).$$

Therefore, $G = \text{J}f(x)^\top$ satisfies condition 2 of Proposition 8.6.3: it is a lex-subgradient of f at x. □

However, we cannot obtain the reverse inclusion so easily: indeed, let $G \in \partial^{\text{lex}} f(x)$. For all $h \in \mathbb{R}^n$,

$$\frac{f(x+\varepsilon h) - f(x)}{\varepsilon} \geq_{\text{lex}} G^\top h.$$

The standard argument would have us take the limit as $\varepsilon \longrightarrow 0$, but lexicographic inequalities are not preserved when taking limits. Example 8.6.13 actually shows that there can be a lot of other elements in $\partial^{\text{lex}} f(x)$.

Example 8.6.13 (Complete lex-subdifferential of the quadratic function)**.** *We instantiate Proposition 8.4.9 by considering* $f\colon x \in \mathbb{R}^n \longmapsto \frac{1}{2}\|x-g\|^2 v \in \mathbb{R}^m$*, with* $v >_{\text{lex}} 0$*. Let* $j_0 = \min\{j \in [m] : v_j > 0\}$*. For any matrix* $M = \begin{pmatrix} M_1 & \cdots & M_m \end{pmatrix}$*, we have*

$$G = (x-g)v^\top + M \in \partial^{\text{lex}} f(x) \quad \iff \quad M_1 = M_2 = \cdots = M_{j_0} = 0.$$

Proof. We seek a characterization of the lex-subgradients of f:

$$\begin{aligned} G \in \partial^{\text{lex}} f(x) &\iff \forall h,\ f(x+h) - f(h) \geq_{\text{lex}} G^\top h \\ &\iff \forall h,\ \left(\|x+h-g\|^2 - \|x-g\|^2\right)\frac{v}{2} \geq_{\text{lex}} G^\top h \\ &\iff \forall h,\ \left(\|h\|^2 + 2(x-g)^\top h\right)\frac{v}{2} \geq_{\text{lex}} G^\top h \end{aligned}$$

If we decompose $G = (x-g)v^\top + M$, this condition simplifies into

$$G \in \partial^{\text{lex}} f(x) \quad \iff \quad \forall h,\ \frac{\|h\|^2 v}{2} \geq_{\text{lex}} M^\top h.$$

We use the columns of $M = \begin{pmatrix} M_1 & \cdots & M_m \end{pmatrix}$ and the components of $v = (v_1, \cdots, v_m)$ to reformulate it as:

$$\forall h,\ \frac{\|h\|^2 v}{2} - M^\top h = \frac{1}{2}\begin{pmatrix} \|h\|_2^2 v_1 \\ \|h\|_2^2 v_2 \\ \vdots \\ \|h\|_2^2 v_n \end{pmatrix} - \begin{pmatrix} M_1^\top h \\ M_2^\top h \\ \vdots \\ M_n^\top h \end{pmatrix} \geq_{\text{lex}} 0. \tag{8.2}$$

The key insight is that if $M_j \neq 0$, by taking $h = \varepsilon M_j$ with small enough $\varepsilon > 0$, we can obtain a negative j-th component. Let us study the different scenarios that may arise:

- If $v_1 < 0$: Not possible since $v \geq_{\text{lex}} 0$
- If $v_1 > 0$: Imposing $M_1 = 0$ is necessary and sufficient for Equation (8.2) to hold
- If $v_1 = 0$: Imposing $M_1 = M_2 = 0$ is necessary but not always sufficient depending on v_2.
  - If $v_2 < 0$: Not possible since $v \geq_{\text{lex}} 0$ and $v_1 = 0$
  - If $v_2 > 0$: Imposing $M_1 = M_2 = 0$ is necessary and sufficient for Equation (8.2) to hold
  - If $v_2 = 0$: Imposing $M_1 = M_2 = M_3 = 0$ is necessary but not always sufficient depending on v_3...

It keeps going like this, and we end up with the following criterion. Let $j_0 = \min\{j \in [m] : v_j > 0\}$ be the index of the first positive component in v. Then

$$G = (x-g)v^\top + M \in \partial^{\text{lex}} f(x) \quad \iff \quad M_1 = M_2 = \cdots = M_{j_0} = 0.$$

□

8.6.4 Lexicographic conjugation

Adapting the definition of the Fenchel conjugate is straightforward.

Definition 8.6.14 (Lex-conjugate)**.** *Let* $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ *be a proper function. Its* lex-conjugate *is the function* $f^{\text{lex}*} : \mathbb{R}^{n\times m} \longrightarrow \overline{\mathbb{R}}^m$ *defined by*

$$\forall G \in \mathbb{R}^{n\times m}, \quad f^{\text{lex}^*}(G) = \operatorname*{lexmax}_{x\in\mathbb{R}^n} \Big(G^\top x - f(x)\Big) \in \overline{\mathbb{R}}^m.$$

Proposition 8.6.15 (Convexity of the lex-conjugate)**.** *Let* $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ *be a proper function. Then its lex-conjugate* $f^{\text{lex}*}$ *is a proper lex-convex function. Since* $\operatorname{dom}(f) \neq \emptyset$*, we deduce that* $f^{\text{lex}*}$ *is proper.*

Proof. The lex-conjugate is a lex-maximum of affine functions of G. By Propositions 8.4.7 and 8.4.11, it is lex-convex. □

Proposition 8.6.16 (Lex-Fenchel-Young inequality)**.** *Let* $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ *be a proper function. For any* $x \in \operatorname{dom}(f)$ *and* $G \in \operatorname{dom}(f^{\text{lex}*})$*, we have*

$$f(x) + f^{\text{lex}*}(G) \geq_{\text{lex}} G^\top x.$$

Proof. By definition of the lex-conjugate, for any $y \in \operatorname{dom}(f)$,

$$f^{\text{lex}*}(G) \geq_{\text{lex}} G^\top y - f(y).$$

In particular, this is true for $y = x$. □

Proposition 8.6.17 (Fenchel-Young condition for lex-subgradient)**.** *Let* $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ *be a proper function. For any* $x \in \operatorname{dom}(f)$ *and* $G \in \operatorname{dom}(f^{\text{lex}*})$*, we have*

$$f(x) + f^{\text{lex}*}(G) = G^\top x \quad \Longleftrightarrow \quad G \in \partial^{\text{lex}} f(x)$$

Proof. We adapt the proof from Udell (2017). First, suppose that $f(x) + f^{\text{lex}*}(G) = G^\top x$. By Proposition 8.6.16, for any $y \in \mathbb{R}^n$,

$$\begin{aligned} f(y) &\geq_{\text{lex}} G^\top y - f^{\text{lex}*}(G) \\ &= G^\top y - \Big(G^\top x - f(x)\Big) \\ &= f(x) + G^\top(y - x). \end{aligned}$$

Now suppose that $G \in \partial^{\text{lex}} f(x)$. By definition of a lex-subgradient, for any $y \in \mathbb{R}^n$,

$$\begin{aligned} f(y) &\geq_{\text{lex}} f(x) + G^\top(y - x) \\ G^\top x - f(x) &\geq_{\text{lex}} G^\top y - f(y). \end{aligned}$$

Taking the lex-maximum over y yields

$$G^\top x - f(x) \geq_{\text{lex}} \operatorname*{lexmax}_{y\in\mathbb{R}^n} \Big(G^\top y - f(y)\Big) = f^{\text{lex}*}(G).$$

By Proposition 8.6.16, this necessarily implies

$$G^\top x - f(x) = f^{\text{lex}*}(G).$$

□

Proposition 8.6.18 (Lex-subgradient of the lex-conjugate)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper function. For any $x \in \mathrm{dom}(f)$ and $G \in \mathrm{dom}(f^{\mathrm{lex}*})$, we have*

$$f(x) + f^{\mathrm{lex}*}(G) = G^\top x \quad \Longrightarrow \quad x \in \partial^{\mathrm{lex}} f^{\mathrm{lex}*}(G)$$

This result relies on a notational trick. Strictly speaking, a lex-subgradient of $f^{\mathrm{lex}*}\colon \mathbb{R}^{n\times m} \longrightarrow \overline{\mathbb{R}}^m$ should be a linear map from $\mathbb{R}^{n\times m}$ to $\mathbb{R}^m$, or (equivalently) a 3-dimensional tensor in $\mathbb{R}^{n\times m\times m}$. But since $x \in \mathrm{dom}(f) \subseteq \mathbb{R}^n$ is just a vector, when we say $x \in \partial^{\mathrm{lex}} f^{\mathrm{lex}*}(G)$, what we really mean is that the linear map $G \longmapsto G^\top x$ is a lex-subgradient of $f^{\mathrm{lex}*}$ at G.

Proof. Suppose $G \in \partial^{\mathrm{lex}} f(x)$. For any $H \in \mathbb{R}^{n\times m}$, we have

$$\begin{aligned} f^{\mathrm{lex}*}(H) = \operatorname*{lexmax}_{y\in\mathbb{R}^n} \Big(H^\top y - f(y)\Big) &\geq_{\mathrm{lex}} H^\top x - f(x) \\ &= H^\top x - G^\top x + (G^\top x - f(x)) \\ &= (H-G)^\top x + f^{\mathrm{lex}*}(G). \end{aligned}$$

□

In standard convex analysis, the reverse implication would be true for closed convex functions, as a consequence of the Fenchel-Moreau biconjugation theorem (Borwein and Lewis 2010, Theorem 4.2.1). But in the lexicographic setting, closed functions are hard to come by. Indeed, even the lex-epigraph of the zero function is $\mathrm{lexepi}(0) = \mathbb{R}^n \times \overline{\mathcal{K}}_{\mathrm{lex}}$, which is not closed.

8.7 The failure of the lexicographic subgradient method

Since a lex-subgradient $G \in \mathbb{R}^{n\times m}$ is a matrix and not a vector, it does not live in the same space as the input variable $x \in \mathbb{R}^n$. Hence, it is not obvious how to translate standard subgradient algorithms for optimization to the lexicographic setting.

8.7.1 Lex-minimizing a local linear surrogate

An interesting remark is that the lex-subgradient can be useful for local search. Suppose we want to find a step h such that $f(x+h)$ is lex-smaller than $f(x)$. Lex-minimizing $h \longmapsto f(x+h)$ is hard, but since $f(x+h) \geq_{\mathrm{lex}} f(x) + G^\top h$, we can try to lex-minimize the linear surrogate $h \longmapsto G^\top h$ instead. This is the goal of the following proposition.

Proposition 8.7.1 (Lex-minimizing a linear function)**.** *Let $G = \begin{pmatrix} G_1 & \cdots & G_m \end{pmatrix} \in \mathbb{R}^{n\times m}$. The lexicographic minimization problem*

$$\operatorname*{lexmin}_{h\in\mathbb{R}^n} \; G^\top h \quad \textit{s.t.} \quad \|h\| \leq 1$$

has the following optimal solution:

$$h^\star = -\frac{G_{j_0}}{\|G_{j_0}\|} \quad \textit{where} \quad j_0 = \min\{j \in [m] : G_j \neq 0\}.$$

Proof. We first note that $G^\top h = (G_1^\top h, ..., G_m^\top h)$. If $H_1 \neq 0$, the best way to lex-minimize $G^\top h$ is to focus on its first component and make it as small as possible. This amounts to choosing $h = -G_1/\|G_1\|$. On the other hand, if $G_1 = 0$ and $G_2 \neq 0$, we cannot change the first component of $G^\top h$, and so we should choose $h = -G_2/\|G_2\|$ to minimize its second component. The same reasoning works for all G_j, which yields the expected result. □

8.7.2 Naive lex-subgradient method

Proposition 8.7.1 tells us in which direction we should look if we are given a starting point. We can use this insight to adapt the standard subgradient method to our setting.

Algorithm 8.7.2 (Naive lex-subgradient method)**.** *Start from an initial guess $x^1 \in \mathbb{R}^n$. Then, for every iteration $k \in \mathbb{N}$ until a stopping criterion is met, repeat the following steps:*

Step 1: *Compute a lex-subgradient $G^k = \begin{pmatrix} G_1^k & \cdots & G_m^k \end{pmatrix} \in \partial^{\text{lex}} f(x^k)$.*

Step 2: *Find the index $j_0^k = \min\{j \in [m] : G_j^k \neq 0\}$ of its first nonzero column and define $g^k = G_{j_0^k}^k$.*

Step 3: *Perform the update $x^{k+1} = x^k - \alpha_k \frac{g^k}{\|g^k\|}$, where α_k is a positive step size.*

Theorem 8.7.3 (Convergence of the naive lex-subgradient method)**.** *Let $f\colon \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex function which admits a lex-minimizer. We apply Algorithm 8.7.2 while assuming that:*

1. *The lex-subgradient norms are uniformly bounded: $\forall j \in [m],\ \exists L_j > 0, \forall k \in \mathbb{N},\ \|G_j^k\| \leq L_j$.*
2. *The step sizes are square summable but not summable: $\sum_{k=1}^{\infty} \alpha_k = +\infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < +\infty$.*
3. *The lex-subgradient uses component m often enough (in terms of total step size): $\sum_{k \in \mathcal{M}} \alpha_k = +\infty$, where $\mathcal{M} = \{k \in \mathbb{N} : j_0^k = m\}$.*

Then, for every $\varepsilon > 0$, there is an index k such that x^k is an ε-approximate lex-minimizer of f.

Before delving into the proof, let us note that this result is not actionable: there is no practical way to check whether $\sum_{k \in \mathcal{M}} \alpha_k = +\infty$, no matter how far we look. To detect these situations, we suggest a heuristic convergence criterion: monitor the partial sums of $\sum_{k \in \mathcal{M}} \alpha_k$, and if they stop growing, something probably went wrong.

Proof. We adapt the proof from the real-valued case, mixing ideas from Bonnans et al. (2006, Theorem 9.3) and Pilanci and Boyd (2021–2022).

First, we exhibit a minimum-approaching direction. Let $\mathcal{X}^\star$ denote the set of lex-minimizers of f, and let $x^\star \in \mathcal{X}^\star$ be any one of them. Using the recursion equation and expanding the square norm, we get:

$$\|x^{k+1} - x^\star\|^2 = \left\|x^k - \alpha_k \frac{g^k}{\|g^k\|} - x^\star\right\|^2 = \|x^k - x^\star\|^2 + \alpha_k^2 - 2\frac{\alpha_k}{\|g^k\|}(g^k)^\top (x^k - x^\star) \tag{8.3}$$

Since $G_{1:j_0^k - 1} = 0$, by Proposition 8.6.2, the vector x^k lex-minimizes $f_{1:j_0^k - 1}$ on $\mathbb{R}^n$. Of course, so does $x^\star$ as a lex-minimizer of f. Subsequently, both vectors belong to the level set

$$\mathcal{Y}_{j_0^k} = \{y \in \mathbb{R}^n \colon f_{1:j_0^k - 1}(y) - f_{1:j_0^k - 1}(x^k) = (G_{1:j_0^k - 1}^k)^\top (y - x^k) = 0\}.$$

By Proposition 8.6.3, this means that $g^k = G_{j_0^k}^k$ is a subgradient at x^k of $f_{j_0^k}$ on $\mathcal{Y}_{j_0^k}$. Combining this with the knowledge that $f_{1:j_0^k}(x^\star) \leq_{\text{lex}} f_{1:j_0^k}(x^k)$, we get:

$$0 \geq f_{j_0^k}(x^\star) - f_{j_0^k}(x^k) \geq (g^k)^\top (x^\star - x^k). \tag{8.4}$$

Plugging this into Equation (8.3) yields

$$\|x^{k+1} - x^\star\|^2 \leq \|x^k - x^\star\|^2 + \alpha_k^2 - 2\frac{\alpha_k}{\|g^k\|}(f_{j_0^k}(x^k) - f_{j_0^k}(x^\star)).$$

We remember that $\|g^k\| = \|G^k_{j_0^k}\| \leq L_{j_0^k} \leq \max_j L_j =: L_{\max}$ and we conclude

$$\|x^{k+1} - x^\star\|^2 \leq \|x^k - x^\star\|^2 + \alpha_k^2 - 2\frac{\alpha_k}{L_{\max}}(f_{j_0^k}(x^k) - f_{j_0^k}(x^\star)). \tag{8.5}$$

Second, we prove the boundedness of the iterates. An easy consequence of Equations (8.4) and (8.6.4) is that

$$\|x^{k+1} - x^\star\|^2 - \|x^k - x^\star\|^2 \leq \alpha_k^2.$$

Summing these inequalities over $l \in [k-1]$ gives us

$$\|x^k - x^\star\|^2 - \|x^1 - x^\star\|^2 \leq \sum_{l<k} \alpha_l^2.$$

Because $\sum_{l=1}^\infty \alpha_l^2 < +\infty$, we conclude that the sequence $\left(\|x^k - x^\star\|^2\right)$ is bounded. It follows that the sequence (x^k) is bounded too.

Third, let us look for a subsequence minimizing the last component. We sum the full Equation (8.5) over $l \in [k-1]$:

$$\|x^k - x^\star\|^2 - \|x^1 - x^\star\|^2 \leq \sum_{l<k} \alpha_l^2 + \frac{2}{L_{\max}} \sum_{l<k} \alpha_l (f_{j_0^l}(x^l) - f_{j_0^l}(x^\star))$$

Bounding $\|x^k - x^\star\| \geq 0$ gives us

$$\frac{2}{L_{\max}} \sum_{l<k} \alpha_l (f_{j_0^l}(x^l) - f_{j_0^l}(x^\star)) \leq r^2 - 0 + \sum_{l<k} \alpha_l^2,$$

where $r = d(x^1, \mathcal{X}^\star)$ is the Euclidean distance between x^1 and the set of lex-minimizers of f. Every term of the sum on the left is nonnegative by Equation (8.4), so we can lower-bound the sum using only the subset $\mathcal{M}$ of indices such that $j_0^l = m$:

$$\begin{aligned}\sum_{l<k} \alpha_l (f_{j_0^l}(x^l) - f_{j_0^l}(x^\star)) &\geq \sum_{l<k, l\in\mathcal{M}} \alpha_l (f_m(x^l) - f_m(x^\star)) \\ &\geq \Big(\sum_{l<k, l\in\mathcal{M}} \alpha_l\Big) \min_{l<k, l\in\mathcal{M}} (f_m(x^l) - f_m(x^\star)).\end{aligned}$$

We thus have

$$0 \leq \min_{l<k, l\in\mathcal{M}} (f_m(x^l) - f_m(x^\star)) \leq \frac{L_{\max}}{2} \frac{r^2 + \sum_{l<k} \alpha_l^2}{\sum_{l<k, l\in\mathcal{M}} \alpha_l}.$$

By hypothesis, $\sum_{l\in\mathcal{M}} \alpha_l = +\infty$ and $\sum_{l\in\mathbb{N}} \alpha_l^2 < +\infty$, so that when $k \longrightarrow +\infty$,

$$\min_{l<k, k\in\mathcal{M}} (f_m(x^l) - f_m(x^\star)) \longrightarrow 0.$$

Fourth and last, we find an approximate lex-minimizer. For every $\varepsilon > 0$, there is an index $k \in \mathcal{M}$ such that $f_m(x^k) \leq f_m(x^\star) + \varepsilon$. But recall that since $k \in \mathcal{M}$, we have $G^k_{1:m-1} = 0$, so that x^k lex-minimizes $f_{1:m-1}$ on $\mathbb{R}^n$ (by Proposition 8.6.2). This shows that $f(x^k) \leq f(x^\star) + \varepsilon \mathbf{e}_m$: in other words, x^k is an ε-approximate lex-minimizer of f. □

Theorem 8.7.3 and its proof underline the pitfalls of this naive algorithm: there are cases where $j_0^k = m$ will never (or rarely) happen, so that the optimization does not succeed. One obvious reason is that, in practice, checking whether $g_j^k \neq 0$ does not make much numerical sense. But that is easily fixable if we replace this equality condition with a norm threshold ε, and use $j_\varepsilon^k = \min\{j \in [m] : \|G_j^k\| \geq \varepsilon\}$ instead of j_0^k.

The deeper issue is that taking a step to improve the second objective may actually set us back with respect to the first, so that we have to go back and fix the optimality that we destroyed. To correct this behavior, we need a way to make sure that steps taken to improve f_j will not degrade any f_i for $i < j$.

8.8 More advanced optimization algorithms

In this section, we propose two alternative algorithms, whose convergence is only conjectured. We leave their detailed investigation for future work, but Section 8.10 displays promising numerical experiments.

8.8.1 Orthogonalized Jacobian method

To overcome the weaknesses of Algorithm 8.7.2, we suggest orthogonalizing the lex-subgradient matrix. This would ensure that steps taken to decrease low-priority objectives do not increase high-priority objectives. But to be sure that a step orthogonal to G_1^k improves f_2 without worsening f_1, we cannot just pick any lex-subgradient column G_1^k. We need to pick *the* gradient of f_1, so that steps orthogonal to it only result in perturbations of order 2 for f_1. This justifies the use of the Jacobian matrix in the following algorithm.

Algorithm 8.8.1 (Orthogonalized Jacobian method)**.** *Start from an initial guess $x^1 \in \mathbb{R}^n$. Then, for every iteration $k \in \mathbb{N}$ until a stopping criterion is met, repeat the following steps:*

Step 1: *Compute the transposed Jacobian matrix $G^k = \mathrm{J}f(x^k)^\top$.*

Step 2: *Orthogonalize it to get $Q^k = \texttt{GramSchmidt}(G^k)$, and define $g^k = \frac{1}{m}\sum_{j=1}^m Q_j^k$.*

Step 3: *Perform the update $x^{k+1} = x^k - \alpha_k \frac{g^k}{\|g^k\|}$, where α_k is a positive step size.*

Note that this orthogonalization trick could also be used for non-convex lexicographic minimization. Alas, Algorithm 8.8.1 does not seem to work on non-differentiable functions. For such cases, we might need a different approach.

8.8.2 Lexicographic cutting planes algorithm

Instead of performing local search with (orthogonalized) gradient steps, we can use cutting planes to construct a polyhedral surrogate objective. Let $\mathcal{C} \subset \mathbb{R}^n$ be a polytope (typically a hyperrectangle) known to contain at least one lex-minimizer of f. Now that we have a working notion of lex-subgradient, generalizing cutting planes to the lexicographic setting is straightforward.

Algorithm 8.8.2 (Method of lex-cutting planes)**.** *Start from an initial guess $x^1 \in \mathcal{C}$. Then, for every iteration $k \in \mathbb{N}$ until a stopping criterion is met, repeat the following steps:*

Step 1: *Compute the objective value* $f(x^k)$ *and a lex-subgradient* $G^k \in \partial^{\text{lex}} f(x^k)$.

Step 2: *Update the approximation* $\ell^k \colon y \in \mathcal{C} \longmapsto \operatorname{lexmax}_{l \in [k} \Big(f(x^l) + (G^l)^\top (y - x^l) \Big)$.

Step 3: *Find its lex-minimizer* $x^{k+1} \in \operatorname*{lexargmin}_{y \in \mathcal{C}} \ell^k(y)$.

The variant we presented is Kelley's basic cutting plane algorithm, but more sophisticated bundle methods can be adapted as well. While we were not able to prove convergence to an approximate lex-minimizer, we did manage to exhibit interesting properties of this method.

Proposition 8.8.3. *The function* ℓ^k *is a lex-lower approximation of* f. *For every* $l \in [k]$, *it satisfies* $f(x^l) = \ell^k(x^l)$ *and* $G^l \in \partial^{\text{lex}} \ell^k(x^l)$.

Proof. Let $y \in \mathbb{R}^n$. For every $l \in [k]$, by choice of $G^l \in \partial^{\text{lex}} f(x^l)$, we have $f(y) \geq_{\text{lex}} f(x^l) + (G^l)^\top (y - x^l)$. By lex-maximizing over $l' \in [k]$, we get

$$f(y) \geq_{\text{lex}} \operatorname*{lexmax}_{l' \in [k} \Big(f(x^{l'}) + (G^{l'})^\top (y - x^{l'}) \Big) = \ell^k(y) \geq_{\text{lex}} f(x^l) + (G^l)^\top (y - x^l),$$

where equality holds for $y = x^l$. □

Proposition 8.8.4. *The sequence of functions* (ℓ^k) *is lex-increasing and each* ℓ^k *is lex-convex.*

Proof. Since ℓ^k is defined as the lex-maximum of a growing family of affine functions, the sequence is increasing. Moreover, Propositions 8.4.7 and 8.4.11 imply that each ℓ^k is lex-convex. □

Proposition 8.8.5. *We can compute a lex-minimizer* x^{k+1} *of* ℓ^k *by solving* m *standard LPs. Furthermore, this lex-minimizer satisfies* $\ell^k(x^{k+1}) \leq_{\text{lex}} f^\star$.

Proof. A lex-minimizer x^{k+1} can be obtained as the solution to a lexicographically-constrained LP:

$$\operatorname*{lexmin}_{(y,v) \in \mathcal{C} \times \overline{\mathbb{R}}^m} v \quad \text{s.t.} \quad \forall l \in [k],\ v \geq_{\text{lex}} f(x^l) + (G^l)^\top (y - x^l).$$

We can solve it using the sequential Algorithm 8.2.1, which amounts to a sequence of standard LPs:

$$\forall j \in [m], \quad v_j^\star = \min_{(y,v_j) \in \mathcal{C} \times \mathbb{R}} v_j \quad \text{s.t.} \quad \begin{cases} v_i^\star \geq f_i(x^l) + (G_i^l)^\top (y - x^l) & \forall i \in [j-1], l \in [k] \\ v_j \geq f_j(x^l) + (G_j^l)^\top (y - x^l) & \forall l \in [k] \end{cases}$$

Furthermore, by Proposition 8.8.3, we have $\ell^k(x^{k+1}) \leq_{\text{lex}} \ell^k(y) \leq_{\text{lex}} f(y)$ for every $y \in \mathbb{R}^n$. In particular, $\ell^k(x^{k+1}) \leq_{\text{lex}} f^\star$. □

8.9 Application to ML

We finally have all the tools we need to integrate lexicographic optimization into ML pipelines.

8.9.1 Lexicographic Fenchel-Young losses

A key ingredient is the framework of Fenchel-Young losses introduced by Blondel, Martins, and Niculae (2020). We follow their exposition very closely to make comparison easier.

Definition 8.9.1 (Regularized lexicographic prediction function)**.** *Let $\Omega : \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex regularization function. Let $\Theta \in \mathbb{R}^{n \times m}$ be an objective matrix. The* lexicographic prediction function *regularized by Ω is defined by*

$$\widehat{y}_\Omega(\Theta) = \operatorname*{lexargmax}_{\mu \in \mathbb{R}^n} \left(\Theta^\top \mu - \Omega(\mu) \right).$$

Computing $\widehat{y}_\Omega$ amounts to solving a lex-concave maximization problem. Whenever Ω is smooth enough, all the algorithms mentioned so far (except the naive lex-subgradient algorithm) can be applied to that problem.

Definition 8.9.2 (Lex-Fenchel-Young loss)**.** *Let $\Omega : \mathbb{R}^n \longrightarrow \overline{\mathbb{R}}^m$ be a proper lex-convex regularization function. Let $\Theta \in \mathbb{R}^{n \times m}$ be an objective matrix and $\bar{y} \in \mathbb{R}^n$ be a ground truth label. The* lex-Fenchel-Young loss *generated by Ω is*

$$\begin{aligned} \mathcal{L}_\Omega^{\text{lex}}(\Theta, \bar{y}) &= \Omega^{\text{lex}*}(\Theta) + \Omega(\bar{y}) - \Theta^\top \bar{y} \\ &= \left[\Theta^\top \widehat{y}_\Omega(\Theta) - \Omega(\widehat{y}_\Omega(\Theta)) \right] - \left[\Theta^\top \bar{y} - \Omega(\bar{y}) \right]. \end{aligned}$$

The properties listed below show why this loss is well suited to regularized lex-optimization problems.

Proposition 8.9.3 (Properties of the lex-Fenchel-Young loss)**.** *The lex-Fenchel-Young loss satisfies:*

1. *Lex-non-negativity: $\mathcal{L}_\Omega^{\text{lex}}(\Theta, \bar{y}) \geq_{\text{lex}} 0$ for any $\Omega \in \mathbb{R}^{n \times m}$ and $\bar{y} \in \mathbb{R}^n$*
2. *Lex-optimality of regularized prediction: $\mathcal{L}_\Omega^{\text{lex}}(\Theta, \widehat{y}_\Omega(\Theta)) = 0$.*
3. *Lex-convexity: $\mathcal{L}_\Omega^{\text{lex}}(\Theta, \bar{y})$ is convex in Θ*
4. *Lex-subgradient: $\widehat{y}_\Omega(\Theta) - \bar{y} \in \partial_\Theta^{\text{lex}} \mathcal{L}_\Omega^{\text{lex}}(\Theta, \bar{y})$.*

Proof. These properties mainly stem from the definitions of lex-conjugates and lex-subgradients:

1. This is implied by Proposition 8.6.16.
2. This is implied by Proposition 8.6.15.
3. This is implied by the second expression of the lex-Fenchel-Young loss.
4. By item 1, $\widehat{y}_\Omega(\Theta)$ achieves equality in the lexicographic Fenchel-Young inequality:

   $$\Omega^{\text{lex}*}(\Theta) + \Omega(\widehat{y}_\Omega(\Theta)) - \Theta^\top \widehat{y}_\Omega(\Theta) = 0.$$

   By Proposition 8.6.18, this implies $\hat{y}_\Omega(\Theta) \in \partial^{\text{lex}} \Omega^{\text{lex}*}(\Theta)$.

   Lastly, by Proposition 8.6.10, we conclude that $\widehat{y}_\Omega(\Theta) - y \in \partial_\Theta^{\text{lex}} \mathcal{L}_\Omega^{\text{lex}}(\Theta, y)$.

□

8.9.2 Limitations

Since the lex-Fenchel-Young loss is constructed from a lex-conjugate, which relies on lex-maximization, it has no reason to be continuous (remember Example 8.4.13), let alone differentiable. For that reason, we should be wary of using the orthogonal Jacobian method (Algorithm 8.8.1) to minimize the loss. Our remaining options are lex-cutting planes (Algorithm 8.8.2) or the sequential approach (Algorithm 8.2.1), neither of which seems compatible with an ML pipeline trained using gradient backpropagation.

Thus, in spite of our efforts, we have not found a principled algorithm that allows training ML pipelines with lexicographic LP layers. The only exception is when the lexicographic LP is preceded by a very simple neural network, namely an affine function, in which case lex-cutting planes can be applied.

But even though the orthogonalized Jacobian require differentiability, we can hope that the lex-Fenchel-Young loss is differentiable almost everywhere (even though it is discontinuous). In this case, using orthogonalized Jacobian steps seems like a reasonable heuristic to try. This hypothesis will be tested in future work.

8.10 Numerical experiments

8.10.1 Simple function

We first illustrate the behavior of our algorithms on a simple test function:

$$f\colon \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2 \longmapsto \begin{pmatrix} (x_1-1)^2 \\ -(x_1-2)^2 - 1 \\ (x_2+1)^2 - 4 \end{pmatrix} \in \mathbb{R}^3.$$

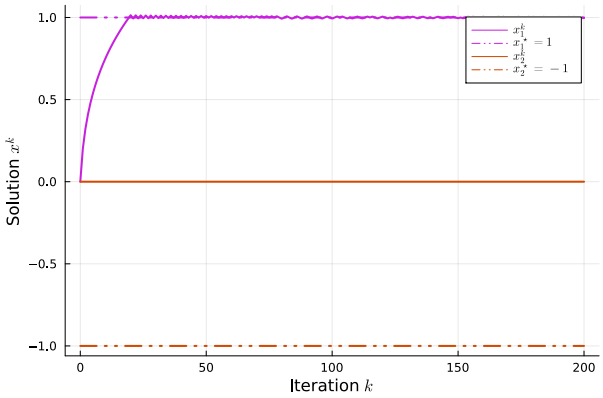
It has a unique lexicographic minimum:

$$x^\star = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \quad \text{and} \quad f^\star = f(x^\star) = \begin{pmatrix} 0 \\ -2 \\ -4 \end{pmatrix}.$$

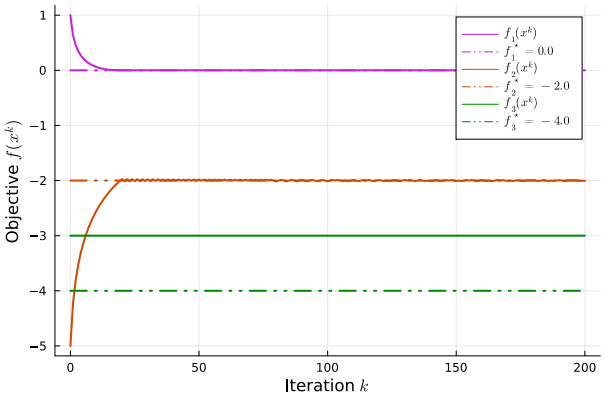
First, we use naive lex-subgradients (Algorithm 8.7.2): the results are displayed on Figure 8.2. As we can see, the solution component x_2 never moves. Indeed, the first two objective components $(x_1 - 1)^2$ and $-(x_1-2)^2 - 1$ are busy fighting each other for attention. Since every step that improves the latter worsens the former, we can never focus on the third objective component $(x_2+1)^2 - 4$. This is very clear on Figure 8.2d: every time $j_0^k = 2$, the next few iterations go back to $j_0^k = 1$ to make things right. Meanwhile, $j_0^k = 3$ never happens, so that the hypotheses of Theorem 8.7.3 are not satisfied. As a result, the optimization is doomed to fail.

Now we move on to the orthogonalized Jacobian method (Algorithm 8.8.1): : the results are displayed on Figure 8.3. Orthogonalization glosses over the conflict between f_1 and f_2 and starts improving f_3 right from the start. Since we are in a differentiable setting, optimization succeeds as planned.

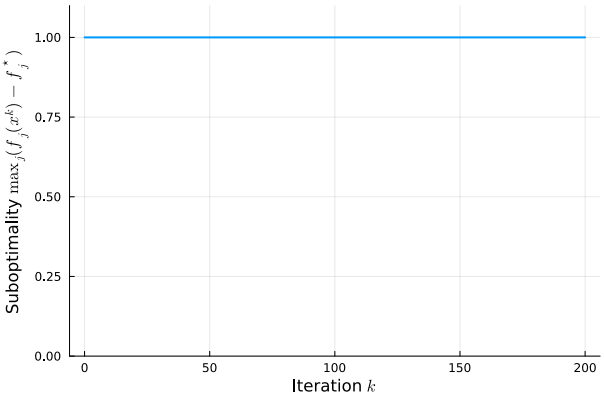
Finally, we turn to lex-cutting planes (Algorithm 8.8.2): the results are displayed on Figure 8.4. Again, all objectives are considered simultaneously, and optimization succeeds. Note that while lex-cutting planes seem to converge in fewer iteration than orthogonal lex-subgradients, each iteration is far more costly because it involves solving several LPs.



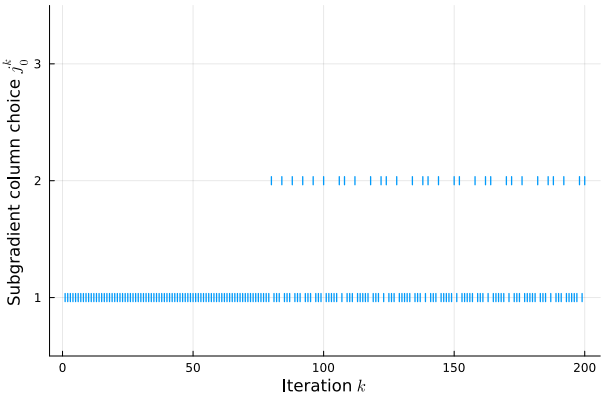
(a) Solution components



(b) Objective components

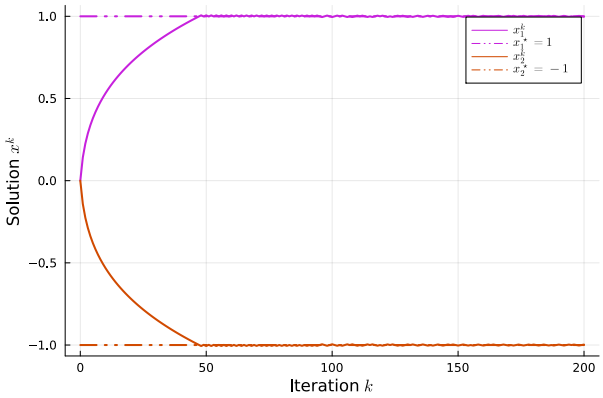


(c) Lex-optimality gap

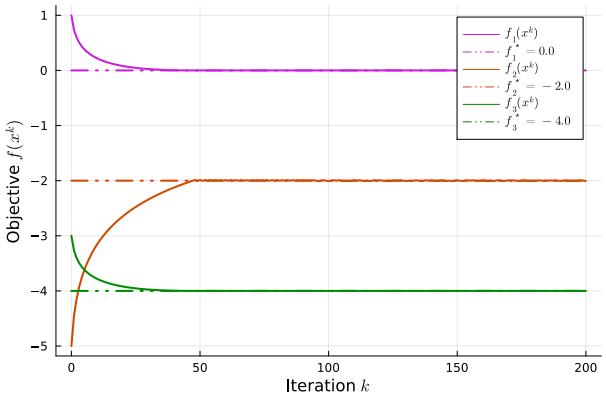


(d) Indices j_0^k chosen from the lex-subgradient

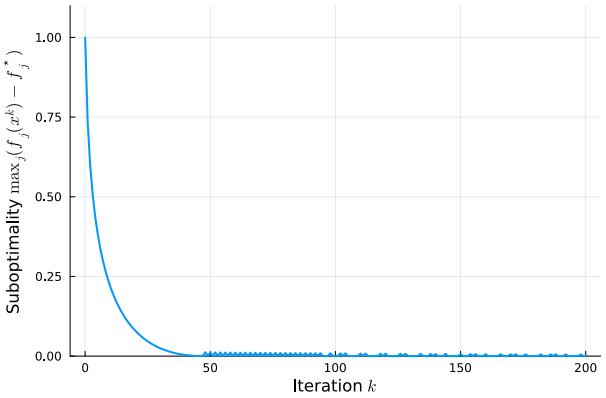
Figure 8.2: Convergence failure of the naive lex-subgradient algorithm



(a) Solution components

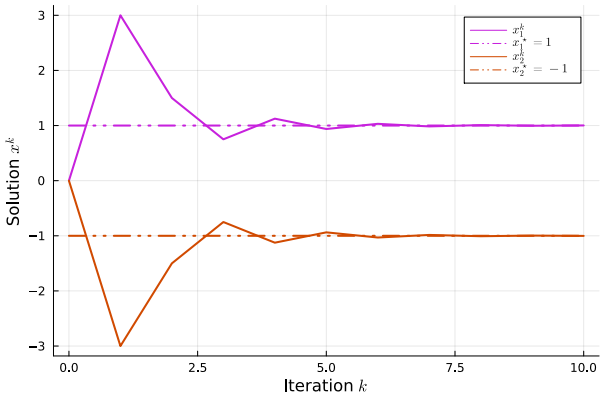


(b) Objective components

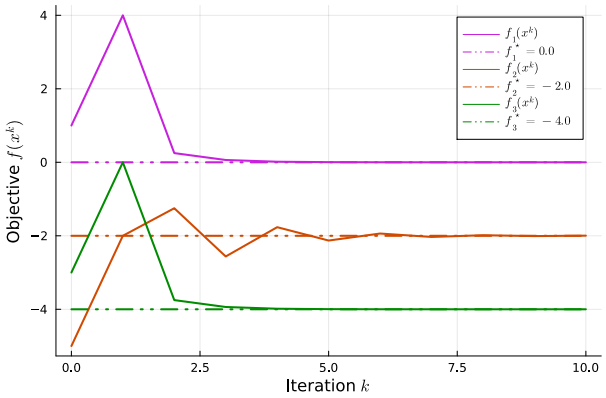


(c) Lex-optimality gap

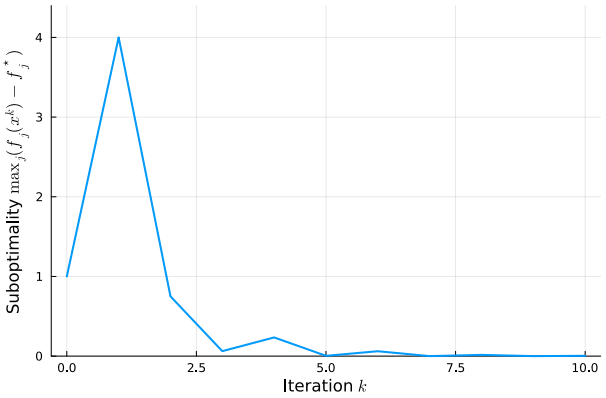
Figure 8.3: Convergence of the orthogonalized Jacobian algorithm



(a) Solution components



(b) Objective components



(c) Lex-optimality gap

Figure 8.4: Convergence of the lex-cutting planes algorithm

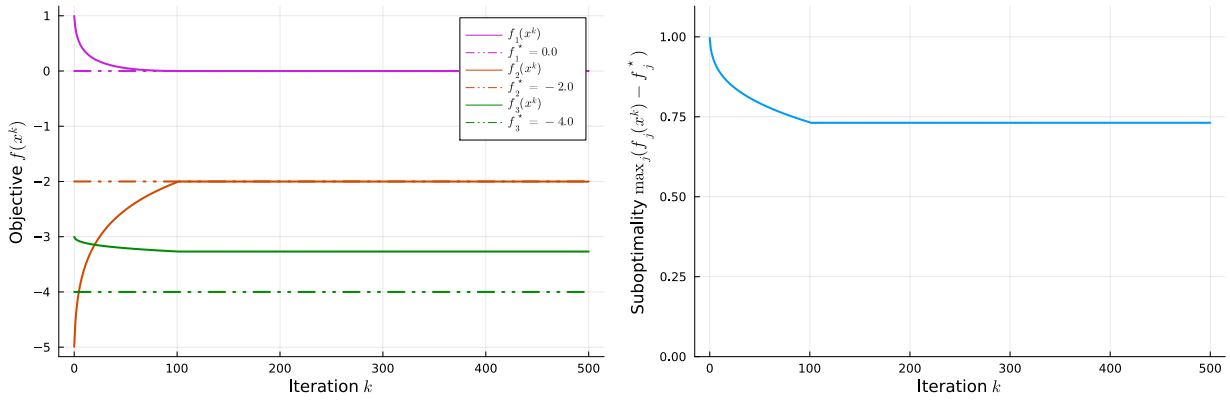
8.10.2 Neural network

We now focus on a more complicated example: a small 3-layer neural network $f = \ell_3 \circ \ell_2 \circ \ell_1$ where

$$\ell_1 = \texttt{Dense(1, 3, r)} \qquad \ell_2 = \texttt{Dense(3, 3, r)} \qquad \ell_3 = \texttt{Dense(3, 2, id)}$$

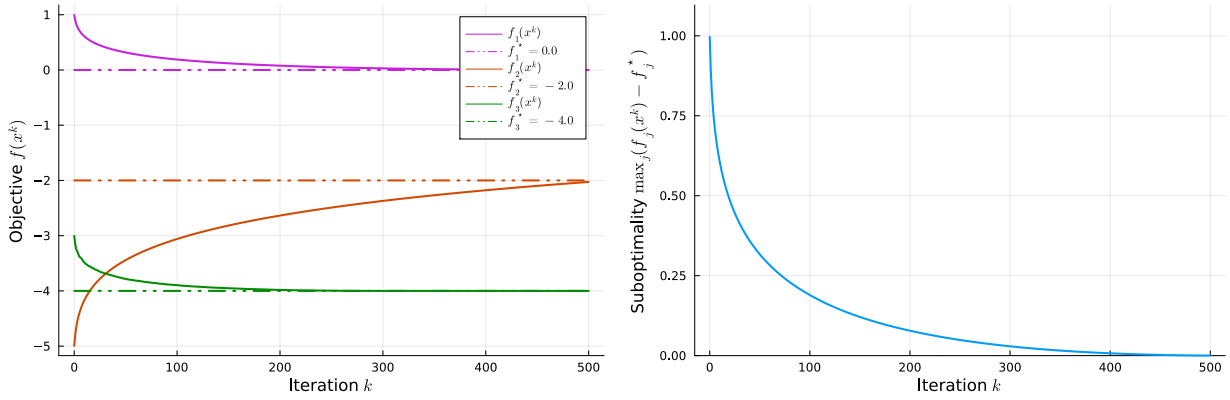
and `r` is a leaky Rectified Linear Unit (ReLU) activation function. We want to lex-minimize the output of the network, but unlike the simple function we used previously, this new objective is not lex-convex. It is not differentiable either, due to the leaky ReLU activations.

But when we compare the lex-subgradient and orthogonalized Jacobian algorithms, we observe the same behavior as before: convergence fails for the former, and succeeds for the latter. This gives us hope that our orthogonalization trick might be useful beyond simple convex settings.



(a) Objective components (b) Lex-optimality gap

Figure 8.5: Convergence failure of the naive lex-subgradient algorithm (neural network)



(a) Objective components (b) Lex-optimality gap

Figure 8.6: Convergence of the orthogonalized Jacobian algorithm (neural network)

Part III

Railway applications

9

Train failure prediction using condition monitoring systems

Oogway: There are no accidents.
Shifu: Yes, I know. You've already said that twice.
Oogway: That was no accident either.
Shifu: Thrice.

Master Oogway & Master Shifu
Kung Fu Panda (2008)

Contents

In this chapter, we propose a model for train failure prediction based on a peculiar stream of condition monitoring data.

9.1 Introduction

Efficient and timely maintenance of the rolling stock is a core challenge for railway companies. Indeed, keeping trains in a functional state is not only essential to guarantee passenger safety, but also to ensure continuity of service.

9.1.1 Predictive maintenance

Jardine, D. Lin, and Banjevic (2006) distinguish several maintenance strategies:

- *Corrective maintenance*, which consists in reacting to breakdowns once they occur.
- *Preventive maintenance*, which involves planning repairs at regular time intervals.
- *Predictive maintenance*, which adapts the repair schedule based on the current and forecasted condition of the system.

While corrective and preventive maintenance are easier to apply, they also have severe drawbacks. On the one hand, relying solely on corrective interventions means that disruptions are bound to happen. On the other hand, periodic repairs may be excessive (and expensive) for a healthy system, but still insufficient for a more vulnerable one. As monitoring data becomes more common and detailed, predictive maintenance is finally within reach, allowing many industries to strike the right balance between maintenance costs and reliability requirements.

9.1.2 Industrial problem

The end goal of our project is being able to answer questions like "what is the probability that this train unit will fail within the next week?" Such information can then be used to optimize the maintenance schedule in near real time, but here we focus on the predictive aspect. To achieve that goal, we were given access to several sources of data over a period of 2 years (2019 – 2021).

Message logs gather the messages sent by each train during and between trips. While some of these messages are fairly normal (*e.g.*, "the engine turned on"), others indicate more or less critical malfunctions (*e.g.*, "the engine only turned on after the second try"). To help with their interpretation, messages are accompanied by *context variables*, a series of categorical or quantitative values that specify the state of various components within the train. We highlight the fact that these messages are recorded continuously, with timestamps that are precise to the second.

Repair logs give us an approximation of the dates at which failures occurred. Indeed, not every failure triggers an event code, which means we may not receive a live warning as it happens. In fact, even specifying what we mean by failure is difficult, because train systems are designed with many duplicated parts in order to keep running safely. For the purposes of this work, we define a failure as something that causes an (unplanned) corrective repair.

Daily activity summaries contain information on the trips performed by a train. In particular, they tell us about the time spent driving, the number of kilometers traveled, and several other features of the train's missions. These help us quantify the amount of stress that each train was subjected to.

Let us stress that the data we work with has a high dimension. We consider a fleet of about 200 train units, and even for the small subsystem we study, they generate more than 10 million messages over the two-year period. Each message has about 50 context variables, and each activity summary

contains around 150 columns. At the same time, our prediction targets are limited in size, with slightly less than 2500 repairs in total. This justifies our search for a model that is both statistically efficient (reasonable number of parameters) and computationally efficient (reasonable CPU and memory footprint).

9.2 Related work

The literature on failure prediction techniques is plentiful and spans several decades. Depending on the field, this topic may appear under several aliases. In engineering, the following keywords are frequently encountered: machine health diagnostics / prognostics, reliability analysis, degradation modeling, estimation of remaining useful life / time to failure, fault detection, *etc.* In healthcare, the notions of survival analysis or longitudinal modeling are more usual.

9.2.1 Stochastic failure prediction

According to Peng, Dong, and Zuo (2010), failure prediction models can either rely on physical models (*e.g.*, differential equations), expert knowledge (translated as a series of "if-then-else" rules) or data-driven approaches. When it comes to complex systems with many components, accurate physical equations do not always exist, and expert knowledge is expensive to acquire and maintain. As a result, here we focus on data-driven approaches.

Salfner, Lenk, and Malek (2010) give a thorough review of data-driven failure prediction, covering both discriminative and generative models. While discriminative models only seek to provide accurate predictions, generative models are concerned with explaining how the system behaves as a whole. In our industrial context, interpretability is a key requirement, which is why we adopt a generative paradigm.

Among the generative family, we can distinguish between deterministic and stochastic models. The latter are of particular interest to us: see Si et al. (2011) for a survey. Treating the system's state as a random variable is a principled way to account for uncertainty, thus avoiding overconfident predictions. In some cases, we have complete observations, which include direct measurements of an appropriate health indicator. But in most cases, the available information is partial, and the state of the system remains hidden. This is the setting we consider here, since message and repair logs do not obviously translate into a measure of degradation.

As underlined by Si et al. (2011), Markovian models play a prominent role in degradation analysis. They make the assumption that knowing the present is enough to predict the future, which simplifies both inference and learning. For the rest of this chapter, we exploit the framework of HMMs, described in Chapter 4.

9.2.2 Influence of exogenous variables

A defining aspect of our railway application is the role played by daily activity summaries, which can be interpreted as exogenous control variables. The literature on survival analysis offers many ways to take these control variables (or covariates) into account: see Klein et al. (2013) for a broad reference. For instance, in the Proportional Hazards model (Cox 1972), the event intensity for each individual is proportional to a population baseline. The individual scaling factor is then expressed using a linear combination of the covariates. Another source of inspiration for us is the Accelerated Failure Time model (Wei 1992), in which covariates modulate the speed at which time flows for each

individual. While standard HMMs do not involve control variables, we have seen in Chapter 4 how to define controlled HMMs that do.

9.2.3 Previous railway-related theses

To conclude this literature review, we mention some doctoral dissertations that focus on failure prediction for railway rolling stock. They were the result of partnerships between SNCF / Alstom and academic labs.

Lair (2011) exploits Piecewise-Deterministic Markov Processes to model degradation for multiple interacting components. He presents an application to air-conditioning units of French regional trains. Sammouri (2014) extracts association rules from message logs sent by Alstom pendolino trains. Dib (2021) explores pattern mining and classical ML to predict failure events from message logs, with applications to French suburban and high-speed trains. The last two theses involve datasets that are very close to the one we study. Both highlight the difficulty of working with real data, and the significant preprocessing work that is necessary. However, to the best of our understanding, neither of them introduces a generative model for the train degradation process.

9.3 Hierarchical degradation model

In this section, we propose a hierarchical model for rolling stock degradation, which leverages all the data we have at our disposal. It is synthesized on Figure 9.1, where arrows represent probabilistic dependencies. Our model is structured like a controlled HMM in discrete time, but the emission distribution is a marked Poisson process in continuous time. This allows us to reconcile the discrete nature of activity summaries with the continuous stream of messages and repair events.

Henceforth, $t \in \mathbb{N}$ corresponds to a calendar day, while $\tau \in \mathbb{R}_+$ is a continuous timestamp. At time t, let u_t denote the controls (train activity), X_t denote the state (health index) and Y_t denote the emissions (event sequence). We now describe each of those layers in more detail, starting from the bottom.

9.3.1 Observations

The bottom layer of our model contains the emissions, or observations. We obtain those by combining messages and repairs, both of which are sequences of events *in continuous time*. In addition to its timestamp $\tau \in \mathbb{R}_+$, each event carries a *mark* $m \in \mathfrak{M}$. For us, marks are tuples $m = (e, c)$ composed of an event type $e \in [E]$ and a vector of contexts $c \in \mathbb{R}^D$ of dimension D (which is empty for repair events). To simplify, we restrict all context values to be categorical and belong to the same finite set $[C]$. Typically, $C = 3$ and we will only allow context values to equal 1, 0 or `missing`.

There are several ways to encode an event sequence into a fixed-size observation vector: the simplest one is to count the number of events of each type within a day. However, the presence of contexts gives rise to many questions. Should we record the average context value? The maximum value? A full histogram? And what about (event, context) pairs? To circumvent these questions, we give up on representing observations as fixed-size vectors. Instead, we consider Y_t as a realization of a marked Poisson process on the interval $[t, t + 1)$. Keeping the whole sequence of events, regardless of its length, allows us to avoid compressing the available information.

To fully specify the marked Poisson process for a given state, we need an intensity value $\lambda(m)$ for each mark $m = (e, c)$. This value is obtained by combining:

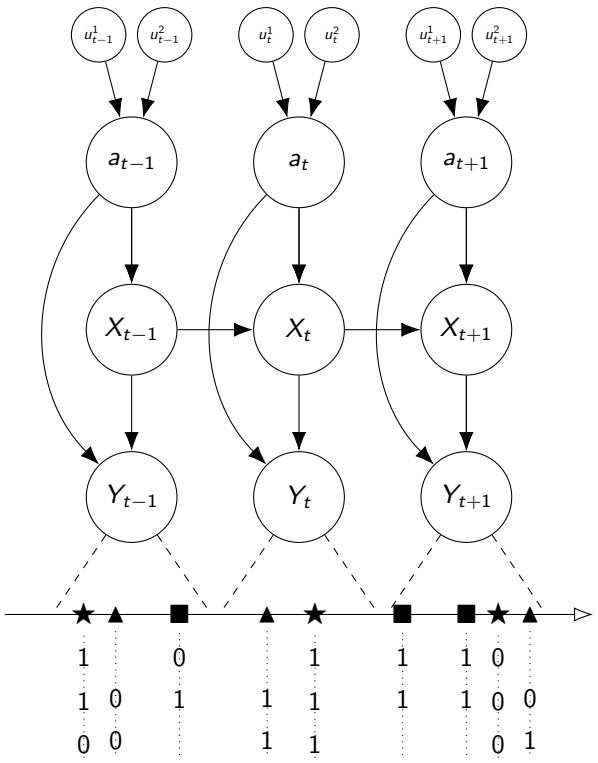


Figure 9.1: Controlled HMM for train failure prediction

- a ground intensity $\lambda \in \mathbb{R}_+$;
- a multinomial distribution $\mathcal{M}(r_0)$ for the event type e, where $r_0 \in [0,1]^E$;
- for each $d \in [D]$, a multinomial distribution $\mathcal{M}(r_d)$ for the context value c_d, where $r_d \in [0,1]^C$.

We further assume that the multinomial distributions are mutually independent, which yields the following expression for the intensity of m:

$$\lambda(m) = \lambda \; r_0(e) \prod_{d=1}^{D} r_d(c_d). \tag{9.1}$$

In the actual railway system we study, the dimension of the context vector c and its possible values depend upon the event type e. Our independence hypothesis simplifies estimation by heavily reducing the number of parameters.

9.3.2 States

The middle layer of our model is related to the health state of the train. We consider a discrete state X_t taking integer values $s \in [S]$, where 1 is as healthy as possible (the nominal state) and S is as degraded as possible (the failure state). The evolution of X_t is a Markov chain is parameterized by the transition matrix $P \in [0,1]^{S \times S}$ and the initial distribution $p \in [0,1]^S$.

As announced, the observations Y_t do not always have the same distribution: the emission parameters listed above actually depend upon the current state X_t of the system. Thus, Equation (9.1) becomes

$$\lambda(m \mid s) = \lambda(s) \; r_0(e \mid s) \prod_{d=1}^{D} r_d(c_d \mid s). \tag{9.2}$$

In order for our model to capture the notion of degradation, we also have to introduce structural constraints. First, the transition matrix must have a specific sparsity pattern that is compatible with a degradation process. A typical example would be

$$P = \begin{pmatrix} \bullet & \bullet & 0 & 0 & 0 \\ 0 & \bullet & \bullet & 0 & 0 \\ 0 & 0 & \bullet & \bullet & 0 \\ 0 & 0 & 0 & \bullet & \bullet \\ \bullet & 0 & 0 & 0 & \bullet \end{pmatrix}$$

where $\bullet$ stands for a nonzero coefficient. This structure only allows stationary transitions from state s to itself, degradation transitions from state s to $s+1$, and a single recovery transition from S to 1. Additionally, we want repair events to be correlated with the health of the system. To that end, we assume that repairs can only occur in the failure state S:

$$r_0(\texttt{repair} \mid s) = 0 \quad \text{for all } s < S.$$

Interestingly, repairs do not have to occur right away when $X_t = S$. Just like in real life, there can be a lapse of time between entering the failure state and undergoing a repair.

9.3.3 Controls

We finally reach the top layer of our model, which deals with the controls. Those are represented as vectors $u_t \in \mathbb{R}^U$ and considered deterministic. Indeed, given the temporal scale of our predictions (a couple of days), it is reasonable to expect that the rolling stock rotations are known for the near future. Since controls come from daily activity summaries, our intuition here is similar to that of the Accelerated Failure Time model. The more active a train is, the faster time flows from its perspective, and the quicker it will be to degrade. So the question becomes: how should we express this acceleration?

Again, interpretability and efficiency requirements invite us to keep the number of parameters as low as possible. For instance, if we allow each control dimension to influence each transition probability, we obtain a number of parameters that scales as $U \times S^2$. Conversely, if we mediate the influence of all control dimensions through a single scalar quantity a_t, then the number of parameters will only scale as $U + S^2$. As its name suggests, the quantity a_t is meant to represent the acceleration of time caused by controls u_t. The case $a_t = 1$ corresponds to the usual flow of time. When $a_t > 1$, the train experiences unusual stress and degrades faster. When $a_t < 1$, the train is unusually idle and degrades slower.

A possible expression for a_t is the following one, inspired by the Proportional Hazards model: $a_t = \exp(w^\top u_t)$, where w is a vector of weights. More generally, one can express $a_t = \varphi_w(u_t)$, where φ_w is a neural network with weights w and nonnegative output.

Once we have computed the acceleration, we need to decide how it modifies the behavior of the state process X_t and observations Y_t. For observations, accelerating time by a_t is tantamount to multiplying the ground intensity by a_t: this yields $\lambda(s, a_t) = a_t \lambda(s)$. For state transitions however, we need an additional idea, which comes from the theory of Markov jump processes (Shelton and Ciardo 2014).

A Markov jump process is the continuous-time equivalent of a Markov chain. It is defined by a matrix $Q \in \mathbb{R}^{S \times S}$ of transition rates: for $i \neq j$, the coefficient Q_{ij} represents the instantaneous rate at which a jump occurs from state i to state j. Meanwhile, we have $Q_{ii} = -\sum j \neq i Q_{ij}$. If Z_t follows a Markov jump process, then its marginal distribution satisfies the following differential equation:

$$\frac{\mathrm{d}\mathbb{P}(Z_t)^\top}{\mathrm{d}t} = \mathbb{P}(Z_t)^\top Q \qquad \Longrightarrow \qquad \mathbb{P}(Z_{t+\tau})^\top = \mathbb{P}(Z_t) \exp(\tau Q)$$

That is why we suggest the following form for the controlled transition matrix: $P(a_t) = \exp(a_t Q)$, where Q is the transition rates matrix associated with a one-day interval. Unfortunately, the matrix exponential implies that even if Q forbids certain transitions, $P(a_t)$ might allow them. Indeed, as soon as Q is irreducible, all coefficients of $\exp(\tau Q)$ will be nonzero. However, provided that the mixing time is long enough, this should not affect the ability of $P(a_t)$ to model a degradation process.

9.3.4 Summary

We wrap up this section with a recap of all the steps in our hierarchical model:

1. The controls u_t are transformed by a neural network φ_w to define a time acceleration a_t:

$$a_t = \varphi_w(u_t)$$

2. The acceleration a_t helps deduce a transition matrix $P(u_t)$ from a transition rates matrix Q:

$$P(u_t) = \exp(a_t Q)$$

3. The transition matrix $P(u_t)$ generates a new state X_t from the previous state X_{t-1}:

$$\mathbb{P}(X_t = s \mid X_{t-1} = s', u_t) = [P(u_t)]_{s's}$$

4. The acceleration a_t and the state $X_t = s$ are combined to compute a new ground intensity $\lambda(s \mid a_t)$:

$$\lambda(s \mid u_t) = a_t \lambda(s)$$

5. The ground intensity $\lambda(s \mid u_t)$ and probabilities p define a marked Poisson process for the observation Y_t on $[t, t+1)$. If $y = \{(\tau^k, m^k) : k \in [K]\}$, then by Equation (4.1),

$$\mathbb{P}(Y_t = y \mid X_t = s, u_t) = \prod_{k=1}^{K} \left(\lambda(s \mid u_t) r_0(e^k|s) \prod_{d=1}^{D} r_d(c_d^k|s) \right) \exp(-\lambda(s \mid u_t))$$

The parameters we want to learn are

$$\theta = (w, p, Q, \lambda, p) \quad \text{with} \quad \begin{cases} w \in \mathbb{R}_+^W & \text{neural network weights} \\ p \in [0,1]^S & \text{initial state distribution} \\ Q \in \mathbb{R}_+^{S \times S} & \text{Markov transition rates} \\ \lambda \in \mathbb{R}_+^S & \text{event rates per state} \\ r \in [0,1]^{(E+D\times C)\times S} & \text{context probabilities per state} \end{cases}$$

These parameters can be estimated with the gradient descent scheme described in Chapter 4. Our package `ControlledHiddenMarkovModels.jl` makes this task much easier, and we plan to use it to retrieve useful insights on the mechanisms behind train failures.

10

Delay propagation on suburban railway networks

A wizard is never late, Frodo Baggins.
Nor is he early. He arrives precisely when
he means to.

Gandalf the Grey
The Lord of the Rings:
The Fellowship of the Ring (2001)

Contents

In this chapter, we propose a new model for train delay propagation, discuss its statistical properties and apply it to real data from the Swiss railway.

This chapter is based upon my unpublished master's thesis, as well as the first version of our paper D. and de Castro (2022).

10.1 Introduction

Railway delays come in two forms: primary and secondary delays. While primary delays are exogenous, and therefore unpredictable, secondary delays are endogenous, caused by interactions between the trips themselves. Indeed, since trips share the same resources, they often share delays too, a phenomenon we call *delay propagation*. This phenomenon is grasped qualitatively by railway experts, but here we seek to quantify it through mathematical modeling.

10.1.1 Delay propagation mechanisms

At its core, delay propagation is about resources. If two trips require the same resource and cannot use it at the same time, then one of them will have to wait until the other has finished. The most important of these resources is infrastructure (track sections, switches, platforms), but rolling stock, crew and passengers also play a significant role. See Chapter 2 for more details.

Aside from resource conflicts, delay propagation is influenced by several other factors. For instance, schedule slack (*i.e.*, the margins incorporated into the train schedule) can contribute to delay absorption. First, at the level of an individual train, travel times are slightly overestimated to help neutralize small perturbations. Second, the buffer interval between trains are also important to prevent delays from propagating endlessly through the schedule. During peak hours on a suburban network, trains follow one another at near maximal frequency. This makes delay absorption difficult, unless some trains are cancelled or rerouted to reduce the load on the network. Conversely, since very few operations take place at night, delays are almost always reset from one day to the next.

Traffic regulators adapt the schedule in real time when something goes wrong. For a suburban network in a densely populated area, the priority is not really punctuality but rather regularity. Thus, the schedule can be significantly modified to minimize delay propagation. Such is the task of the regulators, who have several tools at their disposal: changing the order of trains, the allocation of tracks, cancelling some stops, or even rerouting the trains.

Signals are traffic lights that warn train drivers when another train occupies the track section in front of them, giving them enough time to brake if necessary. However, some drivers may be overly careful and slow down more than they have to. This can amplify delay propagation and induce artificial slowdowns, even long after the primary delay has dissipated.

10.1.2 Industrial problem

Our goal is to design a model with minimal data requirements, which can then be used in various settings. With this in mind, we can only rely on data that is reasonably easy to acquire, namely departure and arrival times at stations (or other checkpoints along the tracks). The theoretical timetable is usually available on the website of the railway company. Realized event times are slightly less accessible, but they can often be queried from a real-time API. On the other hand, obtaining more refined information like GPS feeds is much harder, when it is possible at all.

Since we only work with measurements at stations, we cannot precisely locate a train that does not move. If a train stands still, we will not know where it is nor how late it is until the next station is reached. Given that the distance between stations ranges from a few hundred meters to several tens of kilometers, this can give rise to large positional uncertainties. It also means we do not know what is happening at the microscopic level of track sections and switches. The platform assignment in stations is a black box too.

Because incidents prompt regulators to modify the schedule in real time, there are many inconsistencies between the theoretical timetable and its empirical counterpart. If a train was supposed to stop at a station but was actually cancelled, we will see a database entry with a scheduled arrival and departure time, but no mention of the realized times. The reverse holds for trains that stop where they were not supposed to.

Finally, the lack of information about driver and rolling stock assignments means our knowledge of resource conflicts is very limited. As we will see below, this scarce data is the main motivation for the new approach we introduce. Indeed, existing methods from the literature require a much more detailed knowledge of the system than we possess.

10.2 Related work

In practice, the most basic prediction method used by railway operators is extrapolation. Whenever a train is running 5 minutes late, extrapolation assumes that it will remain 5 minutes late at every stop of its journey. While simple to implement, extrapolation completely overlooks interactions between trains. We now try to give a brief overview of approaches that do take such interactions into account. For a more extensive survey, the reader can refer to Spanninger et al. (2022).

10.2.1 Methods based on the temporal event graph

Many studies on delay prediction rely on a graphical representation of the schedule, which we will call the *temporal event graph* (Kecman and Goverde 2015). Its nodes correspond to train events such as arrivals or departures, and its arcs to precedence relations between events. These precedence relations arise mainly from resource conflicts. Every arc of the event graph is therefore associated with a fixed duration, which acts as a minimum time interval between the two events.

For instance, consider an arc linking the arrival of a train k at a station to its departure from the same station. This arc will have a duration equal to the prescribed dwell time on the platform. Similarly, consider another arc linking the departure of train k_1 from a station to the arrival of the next train k_2 at the same station. This other arc will have a duration depending on the minimum headway time between both trains.

10.2.1.1 Deterministic delay prediction

Let us assume that the temporal event graph describes the schedule perfectly, and that no perturbations occur. Then, starting from a known state of the network, we can compute future event times exactly: all it takes is to consider the event nodes in topological order. For each node, its event time will be the maximum of its scheduled time and all the event times of parent nodes (augmented with the associated arc duration). In simpler terms, an event cannot happen before its scheduled time, and it cannot happen before all its predecessor events have happened and the minimum waiting times have elapsed.

The seminal paper by Goverde (2007) uses max-plus algebra and spectral analysis to analyze the stability of a periodic train schedule. Burdett and Kozan (2014) draw from mathematical scheduling theory to identify the operations affected by a single delay, and thus quantify schedule robustness. The temporal event graph method is generalized by Kecman and Goverde (2015), insofar as running times and dwell times are no longer fixed but computed dynamically using previously-calibrated prediction models.

However, deterministic modeling of the railway system is a very strong assumption which we do not want to make, especially while studying a suburban network where perturbations are the norm. This justifies the addition of stochastic elements to the model, in order to better quantify noise and uncertainty.

10.2.1.2 Stochastic delay prediction

Some papers try to uncover families of probability distributions that are compatible with the delay operations induced by the temporal event graph: sums, maxima and thresholding. The idea is to explicitly compute distributions for future event times. Examples include phase-type distributions (Meester and Muns 2007) or θ-exponential distributions (Büker and Seybold 2012).

Yet, in order to propagate these distributions exactly through the temporal event graph, their number of parameters needs to grow, which often makes approximations necessary. Therefore, a more interesting approach could be to give up on exact formulas and instead rely on established tools for inference in probabilistic graphical models (Koller and Friedman 2009). An illustration of this idea is provided by Corman and Kecman (2018), who interpret the temporal event graph as a linear Gaussian Bayesian Network.

Unfortunately, the previous methods assume that the structure of the temporal event graph is known precisely. As we noted in Section 10.1.2, this assumption is not realistic in our case. The precedence relations arising from logistics (driver, rolling stock) are completely unknown, and even those related to infrastructure use would require more microscopic information than we have. To make matters worse, the temporal event graph changes in real time as the regulators modify the order of trains.

Even Kecman, Corman, and Meng (2015) (authors of numerous studies using the temporal event graph) concede that in the absence of reliable data on resource use, modeling interactions between trains becomes complicated. This is why, on a limited dataset similar to ours, they resort to an independence assumption between trains. Our goal will be to enrich their Markov model by re-introducing interactions in a novel way.

To be fair, automated dependency detection is indeed possible, as demonstrated by Flier et al. (2009). However, their technique relies on a large and consistent dataset for every correlated pair of events. In a suburban network dataset, with strong variability between days and frequent interventions from regulators, it does not seem to apply.

10.2.2 Parametric regression and other methods

Another category of methods uses regression techniques from statistics or ML to build delay prediction systems. Instead of being embedded in a temporal event graph, the interactions between trains can be represented through various input features in a complex parametric model. For their study on freight traffic, Barbour et al. (2018) build a Support Vector Regression model for each origin-destination pair. To predict the arrival time of a specific train, they use the train's individual features but also interaction terms such as its relative priority (compared to other neighboring trains) and various traffic counts (along the route, but also near the origin and destination). Oneto et al. (2018) use one neural network for each train-origin-destination triplet. The input is composed of calendar features, previous delays and event times for the train in question, but also for the other trains that have recently crossed the same section of the network. Arthaud, Lecoeur, and Pierre (2021) combine

the recent Transformer architecture with a graph embedding to allow for massively parallel delay prediction on the whole French railway network.

While they may provide accurate predictions, such black-box techniques lack interpretability due to their complexity. In the present study, our goal is to build a generative model for a physical phenomenon, which is why we will not go down the same path. Furthermore, since the regulation and delay prediction culture at SNCF strongly relies on human expertise, making the model accessible and understandable is a very important part of our work.

A research team at SNCF puts forward a nonparametric delay prediction tool (Chandesris and Chapuis 2018). For any given train, their algorithm considers the event times on the beginning of its journey (up to the present time). Then, the algorithm parses the historical data, looking for other trains with similar journey beginnings. A delay prediction is finally obtained using a weighted mean of the futures of these past trains. The authors mention the similarity measure can be extended to other context elements, which might allow for delay propagation to be included. Nonetheless, their nonparametric approach, does not provide much insight on the generative process behind the delays, which is why we discard it as well.

Finally, we point to an article using disease spreading models to study train delays (Monechi et al. 2018). Without knowledge of the microscopic structure, the authors postulate simple infection rules and recover qualitative properties of the railway system, such as the empirical delay distribution or the emergence of large congested areas. However, their method does not yield quantitative predictions, which makes it ill-suited for our purposes.

10.3 Congestion-based delay model

We now describe our own approach to tackle delay propagation, which was already sketched in the introduction of Chapter 6. It was fuelled by numerous conversations with SNCF railway experts, as well as a two-day observation period in a traffic regulation center near Paris Montparnasse station. To make exposition clearer, we distinguish between the *lateness* of a train, which is an instantaneous measurement (as in "the train is late"), and the notion of *delay*, which is a progressive variation in lateness (as in "the train was delayed").

10.3.1 Qualitative overview

As we will see, our model is based on an analogy with road traffic jams, which seems reasonable given the high frequency of suburban railway traffic.

10.3.1.1 The static event graph

We start by introducing an alternative to the temporal event graph, which we call *static event graph* and denote by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. It can either be constructed by hand, or automatically derived from a historical dataset. Unlike vertices of the temporal event graph, which are associated to a single train at a given point in time, vertices v of the static event graph are shared between all trains and have no temporal aspect.

Each vertex is defined by a tuple *(line, direction, station, event type)*, where the event type can be either departure, arrival or passage. Meanwhile, the edges e do not stand for precedence constraints: they simply represent possible transitions between vertices. The line information is needed when we consider a multi-line network with separate tracks for each. The same goes for

the direction information if both directions have their own track. On the other hand, whenever the infrastructure is shared, we can drop these components from the vertex definition.

The journey of a train can be represented as a path through the static event graph. We measure the lateness at each vertex, and this lateness evolves due to delays encountered along the edges. Importantly, not every edge corresponds to a physical movement. An edge from departure to arrival corresponds to a trip between stations, but an edge from arrival to departure corresponds to a train dwelling at a station. Both types of edges can give rise to delays, through different mechanisms.

10.3.1.2 The notion of congestion

To capture delay propagation, we do not focus on the lateness measured at each vertex (*i.e.*, for each event). Instead, we model the delay suffered along each edge (*i.e.*, between two consecutive events). Our model decomposes this delay into two parts:

- A collective part, which affects several trains and propagates through time and space: we call it the *congestion*.
- An individual part, which affects only one train at a specific instant and location: we call it the *noise*. It can include factors such as driver decisions, passenger behavior, mechanical breakdowns, *etc.*

Note that causality does not play a role in the previous decomposition: whether a given train creates or just experiences the congestion is irrelevant to us.

To understand congestion, picture it as some kind of viscous substance residing on the tracks (or on the station platforms). Depending on how much of it there is, edges of the static event graph are more or less easy to cross. Like a traffic jam or a sound wave, this viscous substance can also move across the network. But crucially, the congestion is a *hidden variable*: we never observe it directly. The only information we have comes from the trains that go through it. The time they need to cross each edge can be considered a noisy version of the underlying congestion.

10.3.2 Quantitative formulation

For every single day n of observations, let us define the following quantities:

- $\overline{Z}^n_{k,v}$ is the scheduled event time for train k at vertex v.
- $Z^n_{k,v}$ is the realized event time of train n at vertex v.
- $X^n_{t,e}$ is the congestion on edge e at time t.
- $\eta^n_{k,e}$ is the noise on edge e at time t for train k.

All trains may not have the same number of events, and all days may not have the same number of trains, but that is perfectly normal.

10.3.2.1 Delay decomposition

Our delay decomposition is expressed as follows: for any train k crossing edge $e = (u, v)$ at time t,

$$\underbrace{Z^n_{k,v} - Z^n_{k,u}}_{\substack{\text{realized duration}\\ \text{of edge } e}} = \underbrace{\overline{Z}^n_{k,v} - \overline{Z}^n_{k,u}}_{\substack{\text{scheduled duration}\\ \text{of edge } e}} + \underbrace{X^n_{t,e}}_{\text{congestion}} + \underbrace{\eta^n_{k,e}}_{\text{noise}} \quad \text{with} \quad t = Z^n_{k,u}$$

We can reformulate this equation in terms of lateness:

$$\underbrace{(Z^n_{k,v} - \overline{Z}^n_{k,v})}_{\substack{\text{lateness at}\\ \text{vertex } v}} = \underbrace{(Z^n_{k,u} - \overline{Z}^n_{k,u})}_{\substack{\text{lateness at}\\ \text{vertex } u}} + \underbrace{X^n_{t,e}}_{\text{congestion}} + \underbrace{\eta^n_{k,e}}_{\text{noise}} \quad \text{with} \quad t = Z^n_{k,u}$$

Finally, if we define $Y^n_{k,e}$ as the delay suffered by train k when crossing edge e at time t, we have:

$$Y^n_{k,e} = (Z^n_{k,v} - \overline{Z}^n_{k,v}) - (Z^n_{k,u} - \overline{Z}^n_{k,u}) = X^n_{t,e} + \eta^n_{k,e} \quad \text{with} \quad t = Z^n_{k,u} \tag{10.1}$$

An essential feature of Equation (10.1) above is the choice of t. The temporal index of the relevant congestion $X^n_{t,e}$ is selected independently of the train number, based solely on the time when the train arrives at the edge.

In Equation (10.1), the noise $\eta^n_{k,e}$ plays an important role, since the congestion is a collective average which does not account for individual phenomena. The simplest way to model it is as a Gaussian:

$$\eta^n_{k,e} \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\nu_e,\ \omega_e^2\right). \tag{10.2}$$

The mean parameter ν_e can be interpreted as systematic delay along edge e, for instance due to an ill-conceived schedule requiring that trains travel faster than they actually can.

As a trivial but important special case, we get the lateness extrapolation model by taking $X^n = 0$ and $\eta^n = 0$: from the present onward, every train keeps its current lateness without adding to it.

10.3.2.2 Evolution of the congestion

The interesting part of our model is the congestion X^n, because it is responsible for delay propagation. Ideally, since the temporal index t from Equation (10.1) can take arbitrary values, we would like X^n_t to evolve in continuous time. However, to simplify statistical analysis, we work in discrete time, and we assume that X^n follows a Vector AutoRegressive (VAR) process with weights matrix $\theta \in \mathbb{R}^{E\times E}$. In other words, for every $t \in \mathbb{N}$, we have

$$X^n_t = \theta X^n_{t-1} + \varepsilon^n_t \quad \text{with} \quad \varepsilon^n_{t,e} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_e^2 I) \tag{10.3}$$

With this approach, it is likely that the discrete time indices t are not on the same scale (expressed in the same unit) as the numerical values of $X^n_{t,e}$ and $Z^n_{k,v}$. Indeed, we can decide to update the congestion every $\Delta = 10$ minutes, while the event times and delays themselves are expressed with arbitrary precision. Then, we need to choose $t = \lfloor Z^n_{k,u}/\Delta \rfloor$ in Equation (10.1), and the total number of steps necessary evolves as $1/\Delta$.

10.3.3 Link with Partially-Observed Vector AutoRegression

When we combine Equations (10.1), (10.2) and (10.3), we obtain a latent variable model that is very close to the Partially-Observed Vector AutoRegression (POVAR) from Chapter 6. We now explain why the central assumptions we made in that chapter hold here too.

10.3.3.1 Sparse dependencies

If two edges e_1 and e_2 are such that $\theta_{e_1,e_2} \neq 0$, then the congestion value X_{t-1,e_1} influences the congestion value X_{t,e_2}. At this point, we should stress that the congestion propagates at a finite speed, and that its spread is dictated by the structure of the static event graph. In particular, if the step duration Δ is not too large, we expect the interactions between steps $t-1$ and t to remain local, *i.e.*, to happen only between edges that are geographically close. This means that the matrix of weights θ is sparse, and that its sparsity pattern is closely related to the adjacency matrix of $\mathcal{G}$. For large networks, sparsity unlocks significant statistical and computational gains, as we saw in Chapter 6.

10.3.3.2 Random observations

In our delay model, the main challenge for estimation is the limited size of the observations. The only congestion values $X^n_{t,e}$ that are directly linked to observations $Y^n_{k,e}$ are those for which at least one train crosses edge e at time t. Subsequently, the number of observed components $X^n_{t,e}$ is directly related to the number of trains circulating on the network, and will represent only a fraction $p \in (0,1)$ of the complete set $[T] \times \mathcal{E}$ of possible couples. This observation pattern can be stored inside a projection matrix Π^n with random binary entries, such that $Y^n = \Pi^n X^n + \eta^n$. The projection matrix contains one row per couple (k,e): on this row, the coefficient in column $(t = \lfloor Z^n_{k,u}/\Delta \rfloor, e)$ is 1, and all the other coefficients are 0.

Importantly, a single congestion value $X^n_{e,t}$ can be observed multiple times by different trains, hence with different noise values. The larger the step duration Δ, the more trains will cross the same edge during a time step, and the more precise our averaged estimate of the latent congestion will be.

To further justify the link with Chapter 6, we claim that the selection of the observed components $X^n_{t,e}$ can reasonably be considered random. First, railway timetables are often large and complex, potentially varying from day to day and subject to last-minute modifications. Therefore, it makes sense to assume that the spatio-temporal locations (t,e) of the observations are randomly sampled.

But more importantly, travel times themselves are not deterministic. If train k reaches edge e slightly later than usual one day, it may face a different congestion value, say $X^n_{t+1,e}$ instead of $X^n_{t,e}$. And because of this, the distribution of Π^n is even influenced by the values of X^n itself, since train k's unusual delay at edge e may have been caused by a traffic jam on edge $e-1$. We capture this phenomenon heuristically, by assuming that the projection matrix Π^n exhibits *spatial and temporal correlations*.

10.3.3.3 Influence of step duration

In light of all we have said, we summarize the various effects of the step duration Δ in Table 10.1.

10.3.4 Statistical estimation

Since we have shown that our delay model is a special case of POVAR, we can apply the estimation algorithm of Chapter 6. Back then, we assumed that each component of X^n was observed at most once in Y^n. Thanks to the use of the Moore-Penrose pseudoinverse, our estimator still works when multiple observations are tied to the same latent variable. The only modification necessary is related to the covariance scaling in Equation (6.9).

	Small Δ	**Large Δ**
Number of time steps T	Large	Small
Structure of the weights θ	Sparse	Dense
Congestion averaging	Rough	Precise

Table 10.1: Effects of the step duration Δ in our delay model

Although we do not provide the proof here, Theorem 6.11 also generalizes to this case, which means we have an upper bound on the estimation error for θ. Roughly speaking, this bound has the following form (if we neglect the logarithmic term):

$$\|\widehat{\theta} - \theta\|_\infty \leq c \left(1 + \frac{\omega^2}{\sigma^2}\right) \frac{s}{p\sqrt{T}}$$

To make sense of this bound, we recall that σ is the scale of the congestion process, while ω is the scale of the noise. The discrete temporal horizon is denoted by T, so we only need reasonable values for the sparsity level s and the (uniform) observation probability p. Let K be the number of trains in a day, and let L be the average number of edges on a train's path in the static event graph. Then a good approximation for the observation probability p would be $\frac{\dim(Y^n)}{\dim(X^n)}$, that is,

$$p \approx \frac{KL}{TE}$$

As for the sparsity level s, it counts the number of edges to which congestion can spread in an interval of duration Δ, starting from a single edge. Let v be the propagation velocity of the congestion, expressed in number edges crossed per time unit. We take d to be the average degree of an edge in the static event graph $\mathcal{G}$. A crude formula for s would be:

$$s \approx d^{v\Delta}$$

10.4 Numerical experiments

Unlike those of the other chapters, the following experiments were implemented in Python. Data preparation was performed using `pandas` (Wes McKinney 2010; Reback et al. 2021), graph structures were represented within `networkx` (Hagberg, Schult, and Swart 2008) while linear optimization problems were modeled using `cvxpy` (Diamond and Boyd 2016; Agrawal, Verschueren, et al. 2018) and solved with the `ECOS` solver (Domahidi, Chu, and Boyd 2013).

10.4.1 Data description and reprocessing

Public transport agencies often make their theoretical transportation plan available (for instance using the General Transit Feed Specification format developed by Google), and many also provide an Application Programming Interface to query real-time traffic information. However, it is much harder to find large historical archives of *realized* event times. One such data set is available on the open data platform Mobility Switzerland[1].

[1] https://opentransportdata.swiss/en/dataset

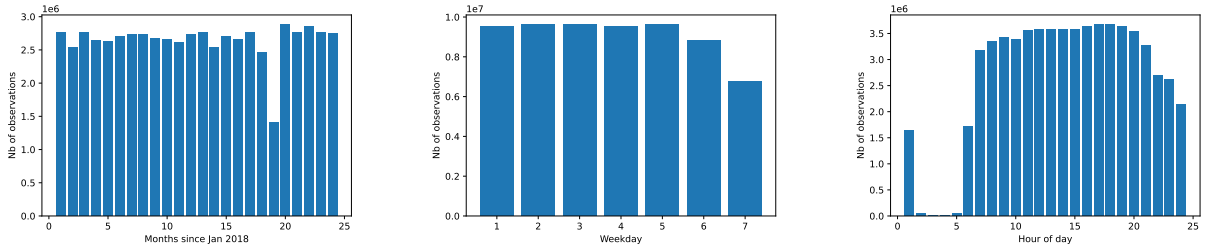


Figure 10.1: Number of observations for each month, weekday and hour on the Zürich tram data set

Starting in January 2018, an increasing number of train arrival and departure times were pulled from the customer information systems of railway companies operating in Switzerland. These event times were then stored into daily CSV files, along with other useful information regarding each train[2]: company, line, trip and stop ID, possible perturbations (like cancelled trips or skipped stops).

Our intuition tells us that a congestion model such as ours best applies to a dense network with frequent trips, for instance that of a large urban or suburban area. As a consequence, we choose to focus on the tramway network of Zürich, operated by *Verkehrsbetriebe Zürich*[3]. We further restrict ourselves to the years 2018 and 2019, since data before 2018 is incomplete and data from 2020 onwards is likely to be affected by the ongoing Covid-19 crisis.

Figure 10.1 gives an overview of the quantity of data available for these two years: these visualizations are important to prepare a homogeneous data set in terms of data quantity. Indeed, if trains are less frequent on a significant portion of the period we study, the congestion will propagate differently and our estimation procedure will be biased.

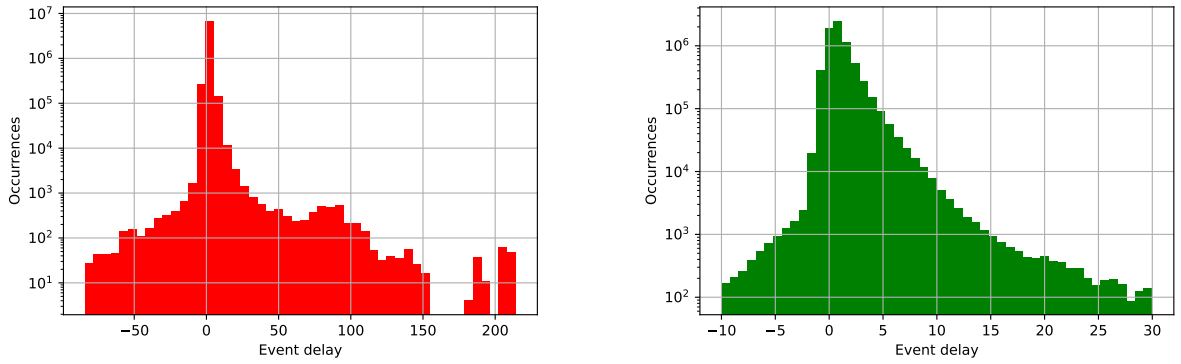
We notice that apart from July 2020, the months are relatively similar to one another. As for the weekdays, Saturdays and Sundays have fewer observations, which is why we exclude them from analysis. Finally, train frequency is zero at night but otherwise relatively constant through the day. Still, we choose to focus on what would be the evening "peak hour" in a typical urban network, that is from 5 PM to 8 PM.

Beyond this initial filtering, we apply a few more preprocessing steps. First, we remove skipped stops, unplanned and cancelled trips. We then remove departures to keep only arrival events. An important step consists in detecting outliers by imposing limits on the minimum and maximum values for edge durations, arrival delays and additional edge delays. This is illustrated on Figure 10.2. It may seem strange to keep slightly negative edge durations or delay values. We made this decision because the planned arrival times are rounded to the minute, while the actual event times are recorded with second-level precision. As a consequence, a train could appear to be a few tens of seconds late or early simply due to rounding phenomena.

The final preprocessing step is to center the data at expectation by removing additional delay averages for each edge. Indeed, for the real process, we suspect that the noise η may not be zero-mean, so this is our way to standardize it.

Then, we need to construct the graph representation $\mathcal{G}$ of our data set, with one vertex per stop and edges corresponding to railway tracks. We could use a network map, but since we want the process to be automated, we seek to build the graph directly from our event data.

[2] https://opentransportdata.swiss/en/dataset/istdaten
[3] https://www.stadt-zuerich.ch/vbz/en/index.html



(a) Histogram of measured arrival delays before cleaning

(b) Histogram of measured arrival delays after cleaning

Figure 10.2: Effect of outlier filtering

To each arrival event, we map the next event for the same train journey, which gives us a collection of station couples, also known as directed edges. Some of these edges are crossed frequently, whereas others are only used a few times over the two-year period. Since the precision of estimation relies on a good approximation of the congestion $X_{t,e}$ on each edge e, we must get rid of these infrequent edges. Indeed, keeping every single edge we obtain (there are over 2500) would result in a much higher dimension for the underlying process, but without sufficient data to exploit it.

Our pruning method consists in selecting the 200 most frequently crossed edges in the complete network, and then retrieving the largest connected component of the resulting subgraph. This component has $|\mathcal{V}| = 78$ nodes and $|\mathcal{E}| = 163$ edges. A graphical representation is given on Figure 10.3a (some edges are denser because of superposition), while the real map can be seen on Figure 10.3b.

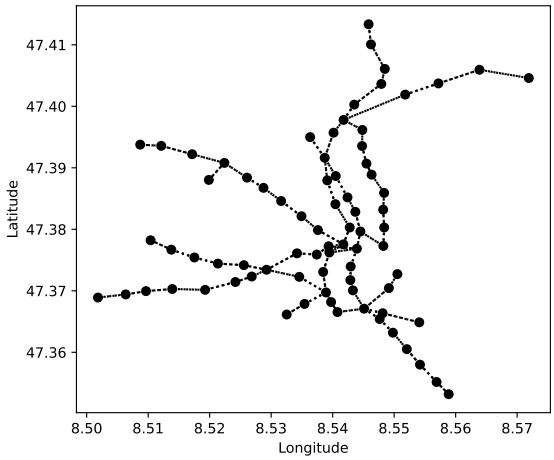
10.4.2 Results

As we mentioned in Chapter 6, cross-validation is difficult to achieve here due to the lack of a standard inference algorithm for hidden congestion values. Given that we don't have access to the sparsity level of the "true" θ here, we test several values of the regularization parameter λ and plot the behavior of the resulting estimator $\widehat{\theta}^{\lambda}$. The main features of interest are presented on Figure 10.4. As expected, the first graph shows the fraction of non-zero coefficients in $\widehat{\theta}^{\lambda}$ decreasing as the penalization λ increases.

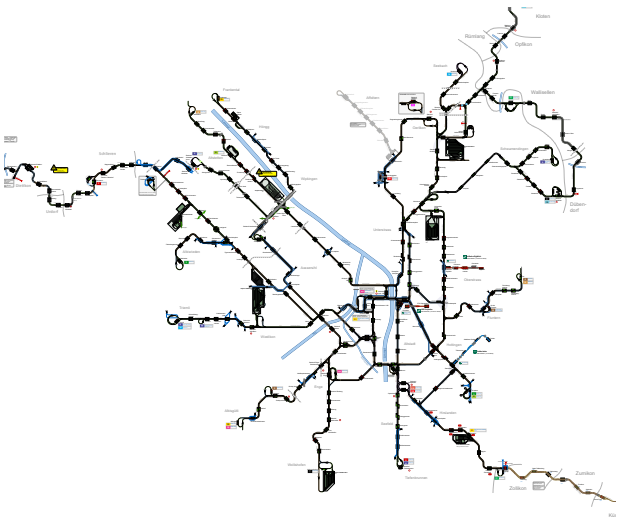
The second graph is more interesting: it depicts the evolution of a quantity characterizing the typical distance at which edges seem to interact, based on the estimated transition matrix. This quantity is computed as a weighted average of distances $d^{\mathcal{G}}_{e_1,e_2}$ between edge couples, each distance being weighed by the absolute value $|\widehat{\theta}^{\lambda}_{e_1,e_2}|$ of the relevant transition coefficient. In other words, our "average interaction distance" is given by the formula

$$\text{AID} = \frac{\sum_{e_1,e_2} |\widehat{\theta}^{\lambda}_{e_1,e_2}| d^{\mathcal{G}}_{e_1,e_2}}{\sum_{e_1,e_2} |\widehat{\theta}^{\lambda}_{e_1,e_2}|}.$$

Since graph distances are only defined between vertices, we need to specify what we mean with $d^{\mathcal{G}}_{e_1,e_2}$.



(a) Map of the Zürich tram network generated from consecutive events



(b) Actual map of the Zürich tram network[a]

[a] https://www.gleisplanweb.eu/Maps/Zuerich.pdf

If $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$, we define it as $d^{\mathcal{G}}_{e_1,e_2} = \min\{d^{\mathcal{G}}_{u_1,u_2}, d^{\mathcal{G}}_{u_2,u_1}\}$. It makes sense because we use the first vertex u of an edge to determine the time step at which a train reaches it. Since G is chosen to be connected, at least one of those two distances $d^{\mathcal{G}}_{u_1,u_2}$ and $d^{\mathcal{G}}_{u_2,u_1}$ will be finite. And since we use mean edge durations as weights, the resulting interaction distance will be expressed in minutes. One could say it measures the distance traveled by the congestion signal during a period of Δ, except that the distance is expressed in minutes (using the average train speed) instead of kilometers.

Our initial intuition for this propagation model is that the network congestion should propagate locally, from one edge to its neighbors, as time flows. And there is one clue on Figure 10.4 that supports this intuition: the fact that interaction distance decreases as the penalization becomes stronger. For small values of λ, the average interaction distance stabilizes around a high value, which is close to the unweighted average of the distances between all pairs of edges (e_1, e_2) (*circa* 12.5 min). In other words, no signal is captured. But as λ rises, we see that the average interaction distance decreases, which suggests that *local effects start to prevail*. This behavior would not be observed if the transition matrix θ were completely independent of the graph structure $\mathcal{G}$.

In addition, when we pick a sufficiently small interval Δ, the average interaction distance seems to stabilize at the end of the curve, whereas it quickly drops to zero for larger intervals. This suggests that picking Δ close to the typical duration of an edge (1.5 min in our data set) may be a good idea.

Of course, there are still things we do not understand, such as the increasing behavior of the curve for $\Delta = 5$ min, or the sudden jump at the end of the one for $\Delta = 7$ min. We assume these must be due to outliers in the data that only appear at a specific sampling frequency. We are also unsure how to compare these curves with one another quantitatively, since each of them captures interactions at a different timescale. If the "true model" was the one with $\Delta = 1$, then each of these curves would roughly correspond to an estimation of θ^{Δ}.

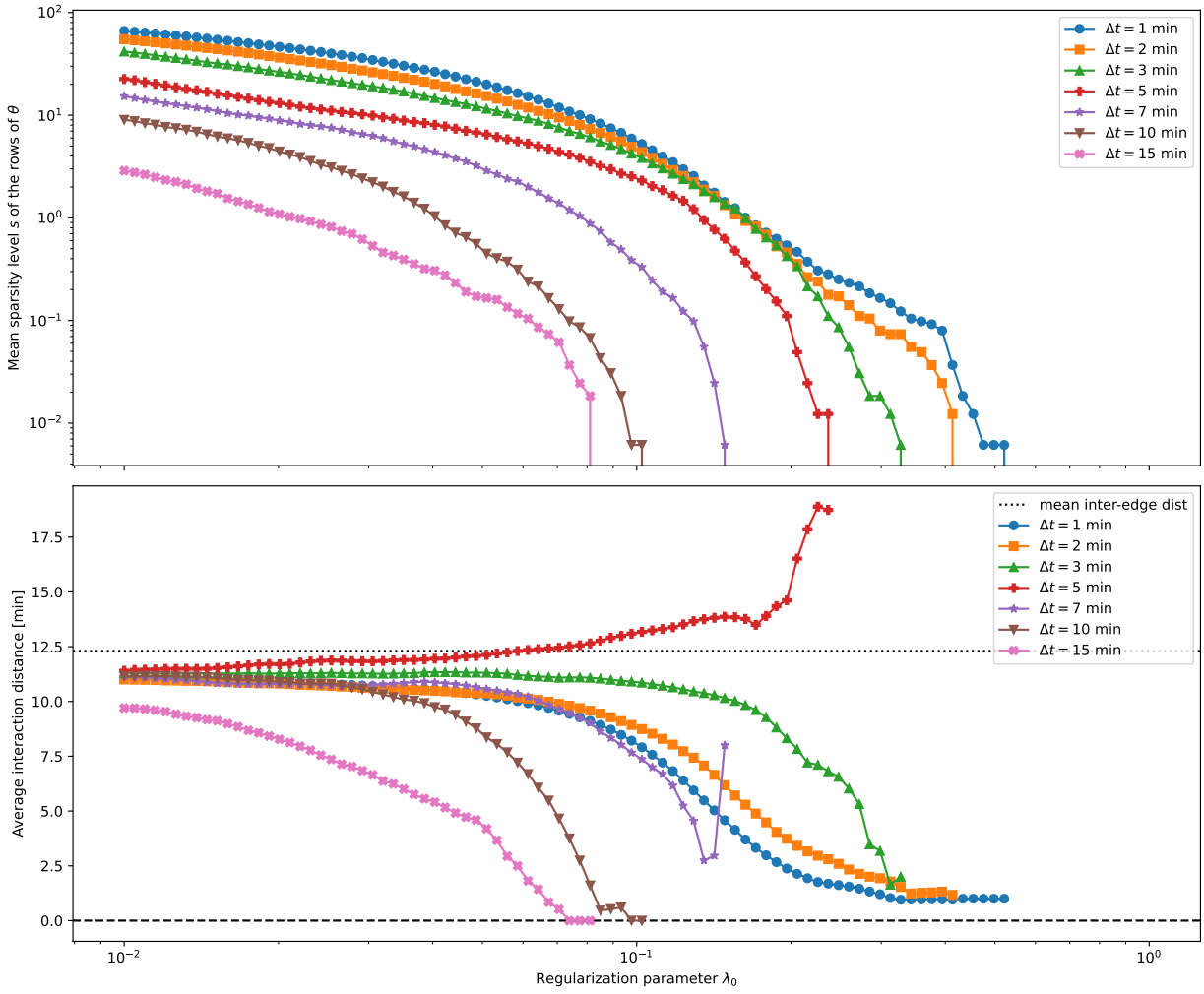


Figure 10.4: Effect of regularization and time discretization interval on some features of the estimate $\widehat{\theta}$

At any rate, we should keep in mind that our procedure is still just a linear Gaussian model, with very little fine-tuning for this specific use case. The fact that we recover a real-world intuition from railway experts is very encouraging, and suggests that we may be on the right path. However, building a more sophisticated predictor that takes into account more network and timetable features would undoubtedly lead to a better understanding of the delay propagation phenomenon.

11

Track allocation for the Flatland challenge

> Parts of the plan were unplanned. That's the plan. I mean, you don't want to overplan a plan.
>
> *Count Olaf*
> A Series of Unfortunate Events – S2E6
> The Vile Village, Part II (2018)

Contents

In this last chapter, we introduce the Flatland challenge and our approach to solving it. Many of the topics that we previously discussed will make an appearance here, as we design an algorithm that draws from both ML and CO.

Parts of this chapter were presented in two talks, one at ROADEF 2022 *(D. and Parmentier 2022) and one at* Journées SMAI-MODE 2022 *(D. 2022).*

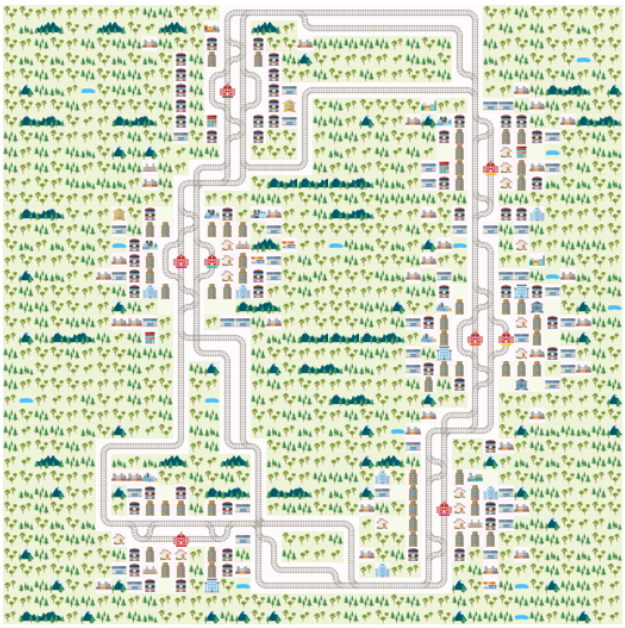


Figure 11.1: Example map generated by the Flatland simulator

11.1 Introduction

In large transportation networks, real-time traffic management is essential to minimize disruptions and maximize punctuality. This is especially true for railway systems, where delays can easily propagate from one train to the next due to very strict infrastructure constraints (see Chapter 10).

11.1.1 A challenge for traffic management

To foster research on this topic, AICrowd launched the Flatland challenge[1] (Mohanty et al. 2020), an international competition centered around a simple railway simulator. This challenge is now organized yearly with the support of *Schweizerische BundesBahnen*, *Deutsche Bahn* and SNCF – its third iteration was just completed. The goal of Flatland is to route multiple trains through a railway network as fast as possible, which requires coordination to prevent unnecessary slowdowns or even deadlocks. The need for real-time rescheduling is justified by stochastic perturbations, which mimic mechanical breakdowns and force other trains to adapt their trajectory.

11.1.1.1 Environment

The Flatland environment is described in detail by its creators (Mohanty et al. 2020). It consists in a two-dimensional grid world of dimension $I \times J$, where each square cell c is located by its Cartesian coordinates (i, j). A cell can either be empty, or it can contain one of several possible rail configurations: straight line, right or left turn, switches. These configurations define the legal transitions from each cell to its neighbors, which also depend on the direction $d \in \mathcal{D} = \{\texttt{North}, \texttt{East}, \texttt{South}, \texttt{West}\}$ of the train. Figure 11.1 displays an example with 8 railway stations.

[1] https://www.aicrowd.com/challenges/flatland-3

Let $\mathcal{C}$ be the set of nonempty cells. We denote by $\mathcal{D}_c^-$ the directions available for a train arriving at cell $c \in \mathcal{C}$, and by $\mathcal{D}_c^+$ the directions available for a train leaving cell $c \in \mathcal{C}$. We denote by $\mathcal{N}_{c,d}$ the set of couples (c', d') that are reachable from (c, d) with a legal transition, that is, the outneighbors of (c, d) – note that this relation is not symmetric. Finally, we also define a smaller subset $\mathcal{S} \subset \mathcal{C}$ of railway station cells.

11.1.1.2 Agents

The environment is populated with a set of A agents which evolve in discrete time during episodes of fixed duration T. At every step $t \in [T]$, we have complete information about the current state of the environment, and all the agents in it. Each agent $a \in [A]$ corresponds to a train, and its trip is defined by the following features:

- its departure cell c_a^{dep} (the railway station where the train enters the network);
- its departure direction d_a^{dep} (the direction with which it appears);
- its arrival cell c_a^{arr} (the railway station where the train exits the network, no matter the direction);
- its earliest departure $t_a^{\min}$ (the train cannot enter the network before that time);
- its latest arrival $t_a^{\max}$ (the train should not exist the network after that time, otherwise it will be considered late).

When entering a new cell, an agent must choose one of five available actions: do nothing, move left, move forward, move right, or stop moving. The chosen action defines whether the agent stays in place or moves to a neighboring cell.

Agents with a speed $s_a = 1$ are able to execute their action directly. On the other hand, agents with a speed $s_a < 1$ must spend $\lceil 1/s_a \rceil$ time steps in the cell before being able to execute the chosen action, if it is still feasible by then. In what follows, we assume that $s_a = 1$ for all agents.

Malfunctions occur following a discrete-time Poisson process with rate λ. Every failure stops an agent for a random duration $\tau \sim \mathcal{U}(\tau_{\min}, \tau_{\max})$; this duration is known as soon as the failure occurs (it can be thought of as the time needed for repairs). In what follows, we neglect failures and focus on the deterministic version of the problem.

11.1.1.3 Costs

In a given episode, the objective is to minimize a sum of costs associated with each agent. If the agent reaches its destination on time, the cost is 0 for this agent. On the other hand, if the agent reaches its destination too late, the cost corresponds to the observed delay. If the agent doesn't reach its destination before the end of the horizon T, we replace the observed delay with the predicted delay (assuming the agent keeps going along its shortest path to the destination, and neglecting interactions). Finally, if the agent never even departs, the previous cost is multiplied by a high penalty. This is meant to discourage train cancellation: otherwise, the easiest way to remove delays would be to cancel all the trains.

11.1.1.4 Evaluation setting

The Flatland challenge is designed to imitate actual railway decision-making, where a lot of time can be allotted to offline planning, whereas online adjustments should happen very quickly. This is reflected in the evaluation method. Every code submission is run for 2 hours on the AICrowd servers, during which the proposed algorithm should complete as many episodes as possible. In each episode, we are granted several minutes of CPU time before the simulation starts, and then a few seconds per time step to indicate the next actions for every agent. An episode ends as soon as one of three conditions is satisfied: (a) all agents have arrived at their destination, (b) the horizon T is reached or (c) one of the agents fails to choose an action in time.

Although our code was not (yet) submitted for evaluation, this protocol helped define our number one criterion for a solution algorithm: the speed of execution.

11.2 Related work

Real-time rescheduling is a common theme in railway research: see Cacchiani et al. (2014) for an overview. However, the Flatland environment provides a very simple approximation of train dynamics, which means the more sophisticated methods from the railway literature may be quite inadequate. The underlying problem is in fact closer to generic Multi-Agent Path Finding, which we surveyed in Chapter 5. Thus, it is not surprising that algorithms designed by MAPF specialists have consistently topped the leaderboard so far (Laurent et al. 2021). In particular, the 2021 and 2022 gold medals were claimed by the team of Li, Z. Chen, Zheng, et al. (2021), competing under the nickname `An_Old_Driver`. Their winning algorithm relies on Sequential MAPF with Large Neighborhood Search. It also involves careful tuning to make the best possible use of the available computation time.

As highlighted by Laurent et al. (2021), RL-based approaches show promise, but they struggle to handle coordination between agents and avoid deadlocks. This is why mixing ML with traditional CO seems like a promising solution: we investigate it now.

11.3 Flatland as a MAPF problem

We now show that the Flatland challenge is a special case of the MAPF framework described in Chapter 5.

11.3.1 Graph building

Let us start by defining the network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on which agents live.

11.3.1.1 Vertices

To construct its vertex set, we combine real vertices and dummy vertices associated with departures or arrivals:

$$\mathcal{V} = \mathcal{V}^{\text{real}} \cup \mathcal{V}^{\text{dep}} \cup \mathcal{V}^{\text{arr}}$$

Real vertices $v = (c, d)$ correspond to the combination of a cell $c \in \mathcal{C}$ and an incoming direction $d \in \mathcal{D}_c^-$:

$$\mathcal{V}^{\text{real}} = \{(c, d) : c \in \mathcal{C}, d \in \mathcal{D}_c^-\}.$$

Departure vertices have the same format, but they are only defined for railway cells:

$$\mathcal{V}^{\text{dep}} = \{(c,d)^{\text{dep}} : c \in \mathcal{S}, d \in \mathcal{D}_c^-\}.$$

Finally, arrival vertices are like departure vertices but without a specific direction:

$$\mathcal{V}^{\text{arr}} = \{(c,\emptyset)^{\text{arr}} : c \in \mathcal{S}\}.$$

Unsurprisingly, we choose the departure and arrival vertices for agent a as follows:

$$v_a^{\text{dep}} = (c_a^{\text{dep}}, d_a^{\text{dep}})^{\text{dep}} \in \mathcal{V}^{\text{dep}} \quad \text{and} \quad v_a^{\text{arr}} = (c_a^{\text{arr}}, \emptyset)^{\text{arr}} \in \mathcal{V}^{\text{arr}}$$

The reason we single out departure and arrival vertices is because we do not know for sure when each agent a will depart or arrive. We only know that departure (resp. arrival) happens at a certain time $t_a^{\text{dep}} \geq t_a^{\text{min}}$ (resp. $t_a^{\text{arr}} \leq t_a^{\text{max}}$). Remember that trains are not yet on the network when they are waiting for departure. Furthermore, they disappear from the network immediately after arrival. During the two intervals $[\![t_a^{\text{min}}, t_a^{\text{dep}} - 1]\!]$ and $[\![t_a^{\text{arr}} + 1, t_a^{\text{max}}]\!]$, we thus need to store agent a in a place where conflicts do not matter.

This justifies the following definitions for incompatibility sets. If a real vertex $v = (c,d) \in \mathcal{V}^{\text{real}}$ is occupied, then no other vertex from the same cell can be:

$$\mathcal{I}_v = \{v' = (c',d') \in \mathcal{V}^{\text{real}} : c' = c\}$$

Meanwhile, dummy departure and arrival vertices $v \in \mathcal{V}^{\text{dep}} \cup \mathcal{V}^{\text{arr}}$ are oblivious to conflicts: we set $\mathcal{I}_v = \emptyset$ in both cases.

11.3.1.2 Edges

We now construct edges similarly:

$$\mathcal{E} = \mathcal{E}^{\text{real}} \cup \mathcal{E}^{\text{dep}} \cup \mathcal{E}^{\text{arr}}$$

Real edges link real vertices to each other or to themselves (since staying in place is always an option):

$$\mathcal{E}^{\text{real}} = \quad \{(c_1,d_1) \to (c_2,d_2) : c_1 \in \mathcal{C}, d_1 \in \mathcal{D}_{c_1}^-, (c_2,d_2) \in \mathcal{N}_{c_1,d_1} \text{ or } (c_2,d_2) = (c_1,d_1)\}$$

Note that when an agent goes from (c_1,d_1) to (c_2,d_2), it can change directions on the cell c_1 by switching from its incoming direction $d_1 \in \mathcal{D}_{c_1}^-$ to an outgoing direction $d_1^+ \in \mathcal{D}_{c_1}^+$. For the transition to be allowed, d_1^+ must be equal to the incoming direction $d_2 \in \mathcal{D}_{c_2}^-$ on the next cell.

Meanwhile, dummy edges link departure and arrival vertices to themselves and to their real counterparts:

$$\mathcal{E}^{\text{dep}} = \{(c,d)^{\text{dep}} \to (c,d)^{\text{dep}} : c \in \mathcal{S}, d \in \mathcal{N}_d^-\} \cup \{(c,d)^{\text{dep}} \to (c,d) : c \in \mathcal{S}, d \in \mathcal{N}_d^-\}$$
$$\mathcal{E}^{\text{arr}} = \{(c,\emptyset)^{\text{arr}} \to (c,\emptyset)^{\text{arr}} : c \in \mathcal{S}, d \in \mathcal{N}_d^-\} \cup \{(c,d) \to (c,\emptyset)^{\text{arr}} : c \in \mathcal{S}, d \in \mathcal{N}_d^-\}$$

We now turn to edge incompatibility sets. When an agent crosses edge e, other agents cannot cross it at the same time, but that is already enforced by vertex incompatibilities. What we really need to forbid is other agents crossing the reverse edge (the so-called swapping conflict). If $e = (c_1,d_1) \to (c_2,d_2)$, then $\text{rev}(e) = (c_2,\text{rev}(d_2)) \to (c_1,\text{rev}(d_1))$, and we define $\mathcal{I}_e = \{\text{rev}(e)\}$.

However, we may want to avoid working with edge incompatibilities, since they require additional bookkeeping. In this case, we can express the same constraint using clever vertex incompatibilities. Given a cell $c \in \mathcal{C}$ and a direction $d \in \mathcal{D}$, we write $c + d$ for the neighbor of c reached after one step along d. The key observation is that a vertex $v \in \mathcal{G}$ in the Flatland graph is associated to a mirror vertex $\text{mirror}(v) = (c + \text{rev}(d), \text{rev}(d))$, which is located "back-to-back" from it. Swapping conflicts are prevented by simply adding $\text{mirror}(v)$ to the incompatibility set $\mathcal{I}_v$.

11.3.2 Objective

As for the objective function, the one from the Flatland challenge is quite complex and involves several cases. To simplify it, we suppose that the horizon T is long enough to let all agents reach their destination. We also put simple weights on the arcs to distinguish those that actually count towards journey duration:

$$\forall e \in \mathcal{E}, \quad \begin{cases} w(e) = 1 & \text{if } e \in \mathcal{E}^{\text{real}} \\ w(e) = 0 & \text{if } e \in \mathcal{E}^{\text{departure}} \cup \mathcal{E}^{\text{arrival}} \end{cases}$$

This leads us to minimizing the following function:

$$\sum_{a=1}^{A} [w(P_a) - (t_a^{\max} - t_a^{\min})]^+$$

where $(\cdot)^+$ denotes the positive part and $w(P_a) = \sum_{e \in P_a} w(e)$ is the total weight of temporal path P_a. Although this function can be easily linearized, to improve readability we will replace it with the flowtime $\sum_{a=1}^{A} w(P_a)$.

11.4 Learning to solve MAPF

We move on to the construction of a hybrid ML-CO pipeline for MAPF, following the blueprint laid out in Chapter 7.

11.4.1 Feature generation

For instance embedding, we draw inspiration from Parmentier (2021a), who suggests a generic and scalable approach. His method relies on identifying relevant "substructures" in our problem, for which we define a set of handcrafted features. These features are then combined using the same weights for each substructure of a given type. Pooling weights like this significantly increases the amount of data per learnable parameter. Of course, using Graph Neural Networks (GNNs) (Chami et al. 2022) for the embedding would also make a lot of sense. However, the coupling of network weights between all substructures would require much more data for training, which may make this approach less practical on small datasets.

In the MAPF case, the most relevant substructures are graph edges and agents. We therefore describe features for each of these types. As stated by several authors (Bengio, Lodi, and Prouvost 2021; Parmentier 2021a), features that stem from the solution of simple optimization problems can be of great interest. In particular, we use independent shortest paths (which are easy to compute) as a basis to construct many relevant indicators.

11.4.1.1 Agent-related features

We propose a first set of features that can be computed for agent a directly:

- Earliest departure and latest arrival of a
- Number of agents departing shortly before or after a

We also propose a second set of features that are based on a set of independent shortest paths:

- Length of the path of a
- Average time step of this path
- Number of waiting steps on this path
- Number of nontrivial switch cells on this path
- Number of conflicts encountered by a
- Number of other agents b crossing its path before/after a
- Total time spent by other agents b on its path before/after a

11.4.1.2 Edge-related features

We propose a first set of features that can be computed for edge $e = (u, v)$ directly:

- Degree of e
- Switch type of the cell containing e
- Existence of the reverse edge
- Existence of parallel edges

We also propose a second set of features that are based on a set of independent shortest paths:

- Number of conflicts encountered on e, u and v
- Number of agents involved in the worst conflict
- Number of agents a whose path includes e, u or v
- Average time step of all crossings of e, u and v

11.4.2 Pipeline based on parallel decomposition

Now that we have an instance embedding, let us turn to the construction of the learning pipeline. Again, we take advice from Parmentier (2021b) as we seek to approximate a hard problem with an easier one. As we saw in Chapter 5, there are two main families of algorithms for MAPF: parallel decomposition and sequential approximation. Alas, the sequential approach cannot be formulated as a standard ILP, and Chapter 8 has shown that lexicographic ILPs are not as easy to differentiate through. That is why we focus on the parallel approach, with the following pipeline:

$$\xrightarrow[\text{instance } x]{\text{MAPF}} \boxed{\text{Encoder}} \xrightarrow[\text{weights } \theta]{\text{Edge}} \boxed{\boxed{\begin{array}{c}\text{Independent}\\ \text{shortest paths}\end{array}}} \xrightarrow[\text{paths } P]{\text{Conflicting}} \boxed{\begin{array}{c}\text{Feasibility}\\ \text{search}\end{array}} \xrightarrow[\text{solution}]{\text{Feasible}} \tag{11.1}$$

In Equation (11.1), we use the encoder to compute our handcrafted features and combine them into modified edge weights θ. These edge weights are then fed to Dijkstra's algorithm, which computes shortest paths for each agent in parallel. Our hope is that finding paths with these new weights will discourage agents from colliding on network bottlenecks. Still, we are likely to encounter conflicts, which is why we need to include a repair procedure. A good candidate is the feasibility search from Chapter 5.

11.4.3 Learning by experience or imitation

As discussed in Chapter 7, there are two ways to learn the encoder parameters: by experience or by imitation. Learning by experience is simpler, because it only requires an evaluator for the cost function (in this case the flowtime).

Learning by imitation is slightly more involved, because we need to precompute good quality solutions to use as targets. A good way to do this would be the double search algorithm from Chapter 5. But since this precomputation happens offline, our CPU budget is virtually unlimited, which means exact algorithms can also be used. Once we have our target paths $\overline{P}$, they can be compared with the paths P created by our pipeline (before the repair step, which is non-differentiable). The applicable loss in this scenario would be the Fenchel-Young loss.

Another approach would be to precompute optimal edge weights θ^* and use them as targets for θ. To achieve this, we turn to standard solution methods for multicommodity flows (Ahuja, Magnanti, and Orlin 1993). One of them is Lagragian relaxation, which basically replaces the no-conflict constraint with a penalization for each edge e in the objective function. The optimal multipliers θ_e^* are then learned using subgradient ascent, and they provide a reasonable target for the edge weights θ_e we want to learn. Indeed, strong duality for LPs tells us that solving independent single-commodity flows with these penalizations yields the same objective value as solving the initial multicommodity flow. Training can then proceed with the SPO+ loss, instead of the Fenchel-Young loss. This idea of learning dual variables was recently leveraged by Dinitz et al. (2021) to speed up graph matching algorithms.

11.4.4 Extension to the stochastic setting

The method described above is still being tested, and preliminary results suggest we will probably need to enrich our set of features. Once we reach satisfying performance, we plan to focus on the stochastic setting, in which train failures are no longer discarded from the simulation.

Parmentier (2021b) tackles a very similar problem in air transport, for which he suggests replacing the deterministic embedding with quantiles of features computed across random simulations. If we have further information on the individual trains and their vulnerability to failures (see Chapter 9), we can also take advantage of that. When put together, these two ideas might suffice to generate resilient schedules, and thus contribute to delay reduction despite unpredictable incidents.

12 Conclusion

> Hurray. Good job, guys. Let's just not come in tomorrow. Let's just take a day. Have you ever tried shawarma?
>
> *Tony Stark*
> Avengers (2012)

Contents

12.1 Summary

The initial motivation for this thesis was to tackle railway planning with data-driven methods. Failure prediction, delay propagation and track allocation are all challenging problems that every railway company must solve efficiently and repeatedly. We thus set out to design new models and algorithms for these tasks, inspired by the latest research in machine learning and combinatorial optimization. However, we soon realized that some key ingredients were missing, both theoretical and numerical.

On the theoretical side, we investigated latent variable processes with randomized observation mechanisms. This made it possible to control the estimation error in our delay propagation framework, based on a few meaningful parameters. We also unified and improved the state of the art for differentiable combinatorial optimization layers. Such layers can be used for structured learning in many scenarios, and we described an application to the Flatland challenge. Finally, our study of lexicographic convex optimization yielded novel insights on an important variant of multiobjective optimization.

On the numerical side, we released 5 Julia packages: `ImplicitDifferentiation.jl`, `InferOpt.jl`, `PointProcesses.jl`, `ControlledHiddenMarkovModels.jl` and `Multi-AgentPathFinding.jl` (plus a number of smaller utilities that we did not describe here, such

as `GridGraphs.jl`, `MAPFBenchmarks.jl` and `Flatland.jl`). Our libraries allow for an easy translation of mathematical ideas into code, and they provide efficient implementations for several important algorithms. We strongly believe that making academic software open source is an essential aspect of ethical and reproducible science.

12.2 Perspectives

For the near future, my research perspectives include writing high performance Julia libraries, as well as investigating further mathematical questions that were raised by the present work. The Julia ecosystem is evolving rapidly, and I intend to contribute to several key areas:

- *Stochastic modeling.* After discussing my work on HMMs, the creator of the reference package `HMMBase.jl` thought I would be the best person to take over its development and maintenance. I plan to extend the breadth of this package, and investigate possible interactions with probabilistic programming frameworks like `Turing.jl`[1] (Ge, Xu, and Ghahramani 2018) or `MeasureTheory.jl`[2] (Scherrer and Schauer 2022).

- *Combinatorial optimization.* The JuliaGraphs organization recruited me to its core development team about a year ago, and there is a lot of work to be done on our flagship package `Graphs.jl`[3] (Fairbanks et al. 2021). On the performance side, it is already a serious competitor to Python's `networkx`[4] (Hagberg, Schult, and Swart 2008), which is why improving it further[5] would bring substantial benefits to the academic community.

- *Automatic differentiation.* The work we initiated on `InferOpt.jl` and `ImplicitDifferentiation.jl` continues, with exciting new applications on the horizon. My invited research stay at the MIT JuliaLab (fall 2022) will also give me the opportunity to set up collaborations on innovative AD approaches.

On the other hand, the theoretical chapters of this thesis still leave some questions unanswered. Regarding latent variable processes, switching to continuous time would make a lot of sense to model railway congestion and other related phenomena. Gaussian processes (C. E. Rasmussen and Williams 2006) seem like an adequate tool, but it is unclear whether they exhibit the same error bounds as their discrete counterparts. Even more uncertain is what happens when the observation mask is entangled with the latent process itself, as is the case for delay propagation. This could give rise to exciting research avenues in high-dimensional statistics.

Regarding combinatorial optimization layers, our probabilistic point of view gives rise to approximations which are not easy to quantify. Can we control the behavior of the relaxed discrete solvers, even when they do not reach an optimal solution? Does our approach generalize well to nonlinear optimization problems, which Blondel, Llinares-López, et al. (2022) have recently started to investigate? How should we tackle other sources of non-differentiability in deep learning? These optimization-related questions will surely fuel several collaborations in the years to come, starting with my MIT colleagues.

[1] https://github.com/TuringLang/Turing.jl
[2] https://github.com/cscherrer/MeasureTheory.jl
[3] https://github.com/JuliaGraphs/Graphs.jl
[4] https://github.com/networkx/networkx
[5] https://github.com/JuliaGraphs/Graphs.jl/issues/128

Finally, my upcoming postdoctoral position at EPFL will focus on epidemic diffusion processes in random graphs. Our goal is to uncover phase transitions using methods from statistical physics and network science. For instance, the feasibility / tractability of source identification probably depends on the connectivity of the graph and the spreading mechanism. Once again, this topic straddles the frontier between machine learning and combinatorial optimization, which is right where I want to be.

Appendices

A Useful lemmas

Sometimes, to solve a case, one must first solve another.

Sherlock Holmes
Sherlock – S4E0
The Abominable Bride (2016)

A.1 Linear algebra

The following set of results will sometimes be used in matrix calculations without explicit justifications.

Lemma A.1.1 (Weyl's inequality)**.** *Let A and B be two $n \times n$ symmetric matrices. Then for all i we have:*

$$\lambda_i(A) + \lambda_n(B) \leq \lambda_i(A+B) \leq \lambda_i(A) + \lambda_1(B).$$

In particular,

$$\lambda_{\min}(A) + \lambda_{\min}(B) \leq \lambda_{\min}(A+B).$$

Proof. See Horn and C. R. Johnson (2012, Theorem 4.3.1). □

Lemma A.1.2 (Ostrowski)**.** *Let S and A be two $n \times n$ matrices with S symmetric. For all i, there is a real number $r_i \in [\varsigma_{\min}(A)^2, \varsigma_{\max}(A)^2]$ such that $\lambda_i(ASA') = r_i\lambda_i(S)$, where $\varsigma_{\min}$ (resp. $\varsigma_{\max}$) denotes the minimum (resp. maximum) singular value.*

Proof. See Horn and C. R. Johnson (2012, Theorem 4.5.9 and Corollary 4.5.11) □

Lemma A.1.3 (Singular values of the Kronecker product)**.** *Let A and B be two matrices. Then*

$$\|A \otimes B\|_2 \leq \|A\|_2 \|B\|_2.$$

Proof. See Horn and C. R. Johnson (1991, Theorem 4.2.15). □

Lemma A.1.4. *For any two matrices A and B, we have:*

$$\|AB\|_F \leq \min\left\{\|A\|_2\|B\|_F, \|A\|_F\|B\|_2\right\}$$

Proof. The Loewner order on symmetric matrices satisfies the following properties:

$$\begin{aligned}\forall (P,Q) \in \mathcal{S}_n(\mathbb{R}), \forall R, \quad P \preceq Q &\implies R'PR \preceq R'QR \\ \forall (P,Q) \in \mathcal{S}_n(\mathbb{R}), \quad P \preceq Q &\implies \mathrm{Tr}(P) \leq \mathrm{Tr}(Q).\end{aligned}$$

The first inequality is true because if x is a vector, $x'R'(Q-P)Rx = (Rx)'(Q-P)(Rx) \geq 0$ due to the Loewner positivity of $Q-P$. The second inequality can be directly deduced from the relation between the spectra of P and Q. Therefore, since $A'A$ is symmetric,

$$B'A'AB \leq \lambda_{\max}(A'A)B'B$$

which implies

$$\|AB\|_F^2 = \mathrm{Tr}(B'A'AB) \leq \lambda_{\max}(AA')\,\mathrm{Tr}(B'B) = \|A\|_2^2\|B\|_F^2.$$

The proof for the other inequality is identical. □

Lemma A.1.5. *Let A and B be two matrices with compatible sizes: then*

$$\|AB\|_{\max} \leq \min\{\|A\|_\infty \|B\|_{\max}, \|A\|_{\max}\|B\|_1\}.$$

Proof.

$$\|AB\|_{\max} = \max_{i,j} |(AB)_{i,j}| = \max_{i,j} \left| \sum_k A_{i,k}B_{k,j} \right|$$

We easily deduce:

$$\|AB\|_{\max} \leq \max_i \left| \sum_k A_{i,k} \right| \times \|B\|_{\max} = \|A\|_\infty \|B\|_{\max}$$

$$\|AB\|_{\max} \leq \|A\|_{\max} \times \max_j \left| \sum_k B_{k,j} \right| = \|A\|_{\max}\|B\|_1$$

□

A.2 Statistics

Lemma A.2.1 (Fano's method)**.** *Let $\theta_0, ..., \theta_M$ be $M+1$ parameters that are 2τ-separated w.r.t. a distance d*

$$\forall i \neq j, \quad d(\theta_i, \theta_j) \geq 2\tau$$

and such that the average KL divergence between $\mathbb{P}_{\theta_i}$ and $\mathbb{P}_{\theta_0}$ is small enough

$$\frac{1}{M+1}\sum_{i=1}^{M} \mathrm{KL}\left\{\mathbb{P}_{\theta_i} \parallel \mathbb{P}_{\theta_0}\right\} \leq \alpha \log M \quad \textit{with} \quad 0 < \alpha < 1 \tag{A.1}$$

Then the minimax probability of an error at threshold τ satisfies:

$$\inf_{\widehat{\theta}} \sup_{\theta \in \Theta_s} \mathbb{P}_\theta \left[d\left(\widehat{\theta}, \theta\right) \geq \tau \right] \geq \frac{\log(M+1) - \log 2}{\log M} - \alpha.$$

Proof. See Tsybakov (2009, Section 2.2 + Corollary 2.6). In particular, since $M \mapsto \frac{\log(M+1)-\log 2}{\log M}$ is increasing, setting $\alpha = \frac{\log(3)-\log(2)}{2\log(2)} \geq 1/2$ is enough to obtain a minimax risk greater than α, as soon as $M \geq 3$. □

Lemma A.2.2 (Chain rule for KL divergence)**.** *If $\mathbb{P}_0$ and $\mathbb{P}_1$ are probability densities on a product space $\mathcal{X} \times \mathcal{Y}$ with $\mathcal{X}$ discrete, then:*

$$\mathrm{KL}\left\{\mathbb{P}_0[X,Y] \parallel \mathbb{P}_1[X,Y]\right\} = \mathrm{KL}\left\{\mathbb{P}_0[X] \parallel \mathbb{P}_1[X]\right\} + \mathbb{E}_X\left[\mathrm{KL}\left\{\mathbb{P}_0[Y|X] \parallel \mathbb{P}_1[Y|X]\right\}\right].$$

Proof. See Cover and Thomas (2006, Theorem 2.5.3). □

Lemma A.2.3 (KL divergence between Gaussians)**.** *The KL divergence between two multivariate Gaussian distributions $\mathbb{P}_0 = \mathcal{N}(\mu_0, \Sigma_0)$ and $\mathbb{P}_1 = \mathcal{N}(\mu_1, \Sigma_1)$ of dimension n is*

$$\mathrm{KL}\left\{\mathbb{P}_0 \parallel \mathbb{P}_1\right\} = \frac{1}{2}\left(\mathrm{Tr}(\Sigma_0\Sigma_1^{-1}) + (\mu_1-\mu_0)'\Sigma_1^{-1}(\mu_1-\mu_0) - n + \mathrm{logdet}(\Sigma_1\Sigma_0^{-1})\right).$$

Proof. See Duchi (2007, page 13). □

Lemma A.2.4 (KL divergence between close Gaussians)**.** *Let Δ be a symmetric matrix of size n such that $\lambda_{\min}(\Delta) > -1$, and let M be a rectangular matrix such that $MM' \succ 0$. Then the KL divergence between*

$$\mathbb{P}_1 = \mathcal{N}(\mu, M(I+\Delta)M') \quad \textit{and} \quad \mathbb{P}_0 = \mathcal{N}(\mu, MM')$$

satisfies

$$\mathrm{KL}\left\{\mathbb{P}_1 \parallel \mathbb{P}_0\right\} \leq \frac{\|\Delta\|_F^2}{2(1+\lambda_{\min}(\Delta))}.$$

Proof. From Lemma A.2.3 (beware of the switch between $\mathbb{P}_0$ and $\mathbb{P}_1$) we get:

$$\begin{aligned}
\mathrm{KL}\left\{\mathbb{P}_1 \parallel \mathbb{P}_0\right\} &= \frac{1}{2}\left(\mathrm{Tr}(\Sigma_1\Sigma_0^{-1}) + (\mu_0-\mu_1)'\Sigma_0^{-1}(\mu_0-\mu_1) - n + \mathrm{logdet}(\Sigma_0\Sigma_1^{-1})\right) \\
&= \frac{1}{2}\left(\mathrm{Tr}(M(I+\Delta)M^{-1}) - n - \mathrm{logdet}(M(I+\Delta)M^{-1})\right) \\
&= \frac{1}{2}\left(\mathrm{Tr}(\Delta) - \mathrm{logdet}(I+\Delta)\right).
\end{aligned}$$

As it happens, for small deviations from the identity, the log-determinant is almost equal to the trace. Indeed, since

$$\forall x > -1, \quad \log(1+x) \geq \frac{x}{1+x},$$

we have

$$\begin{aligned}
\mathrm{Tr}(\Delta) - \mathrm{logdet}(I+\Delta) &= \sum_{k=1}^n \lambda_k(\Delta) - \sum_{k=1}^n \log(1+\lambda_k(\Delta)) \\
&\leq \sum_{k=1}^n \lambda_k(\Delta) - \sum_{k=1}^n \frac{\lambda_k(\Delta)}{1+\lambda_k(\Delta)} \\
&= \sum_{k=1}^n \frac{\lambda_k(\Delta)^2}{1+\lambda_k(\Delta)} \leq \frac{1}{\min_k(1+\lambda_k(\Delta))}\sum_{k=1}^n \lambda_k(\Delta)^2 \\
&= \frac{\|\Delta\|_F^2}{1+\lambda_{\min}(\Delta)}.
\end{aligned}$$

□

Lemma A.2.5 (Chernoff inequality for Bernoulli variables)**.** *Let (X_t) be sequence of independent $\mathcal{B}(p)$ variables. Their average satisfies*

$$\forall u \in [0,1], \quad \mathbb{P}\left(\left|\frac{1}{T}\sum_{t=1}^{T} X_t - p\right| \geq up\right) \leq c_1 \exp\left(-c_2 u^2 T p\right).$$

Proof. See Dubhashi and Panconesi (2009, Theorem 1.1). □

Lemma A.2.6 (Doeblin condition and mixing time)**.** *Let (X_t) be an irreducible aperiodic Markov chain with state space $\mathcal{X}$, transition matrix P and stationary distribution μ. Suppose that (X_t) satisfies the Doeblin condition:*

$$\exists r \in \mathbb{N}, \exists \delta > 0, \forall (x,y) \in \mathcal{X}^2, \quad P^r(x,y) \geq \delta \mu(y).$$

Then the mixing time of X_t, defined as

$$t_{\text{mix}}(\epsilon) = \min\left\{t \in \mathbb{N} : \max_{x \in \mathcal{X}} \left\|P^t(x,\cdot) - \mu\right\|_{\text{TV}} \leq \epsilon\right\},$$

satisfies:

$$t_{\text{mix}}(\epsilon) \geq r\left(1 + \frac{\log \frac{1}{\epsilon}}{\log \frac{1}{1-\delta}}\right).$$

Proof. The proof of Levin et al. (2017, Theorem 5.4) shows that with our assumptions,

$$\forall x \in \mathcal{X}, \quad \left\|P^t(x,\cdot) - \mu\right\|_{\text{TV}} \leq (1-\delta)^{\lfloor t/r \rfloor}.$$

From which we can deduce a sufficient condition for ϵ-mixing:

$$(1-\delta)^{\lfloor t/r \rfloor} \leq \epsilon \quad \Longleftrightarrow \quad \left\lfloor \frac{t}{r} \right\rfloor \geq \frac{\log(\epsilon)}{\log(1-\delta)} \quad \Longleftarrow \quad \frac{t}{r} - 1 \geq \frac{\log \frac{1}{\epsilon}}{\log \frac{1}{1-\delta}}.$$

The result follows easily. □

Lemma A.2.7 (Chernoff inequality for Markov chains)**.** *Let (X_t) be an ergodic stationary Markov chain with finite state space $\mathcal{X}$. We consider a function $f : \mathcal{X} \to \mathbb{R}$ such that $\mathbb{E}[f(X_t)] = \mu$. Then*

$$\forall u \in [0,1], \quad \mathbb{P}\left(\left|\frac{1}{T}\sum_{t=1}^{T} X_t - \mu\right| \geq u\mu\right) \leq c_1 \exp\left(-c_2 \frac{u^2 T \mu}{t_{\text{mix}}(1/8)}\right)$$

Proof. See Chung et al. (2012, Theorem 3) □

Lemma A.2.8 (Chernoff inequality for Markov chains under Doeblin condition)**.** *Under the hypotheses of the previous two Lemmas (A.2.6 and A.2.7), if the parameters r and δ in the Doeblin condition are constants, then we have:*

$$\forall u \in [0,1], \quad \mathbb{P}\left(\left|\frac{1}{T}\sum_{t=1}^{T} X_t - \mu\right| \geq u\mu\right) \leq c_1 \exp\left(-c_2 u^2 T \mu\right)$$

Proof. By Lemma A.2.6, since r and δ are constants, the $\frac{1}{8}$-mixing time of (X_t) can be bounded by a constant

$$t_{\text{mix}}(1/8) \leq r\left(1 + \frac{\log(8)}{\log \frac{1}{1-\delta}}\right) \leq c_3,$$

which we merge with the c_2 inside the exponential of Lemma A.2.7. □

Lemma A.2.9 (Gilbert-Varshamov). *Let $\mathcal{H} = \{0,1\}^d$ be the d-dimensional binary hypercube. If $d \geq 8$, there exists a pruned subset $\mathcal{K} \subset \mathcal{H}$ such that*

$$\forall (x,y) \in \mathcal{K},\ \|x-y\|_1 \geq \frac{d}{8} \quad \textit{and} \quad |\mathcal{K}| \geq 2^{d/8}.$$

Proof. See Tsybakov (2009, Lemma 2.9) □

Lemma A.2.10 (Hanson-Wright inequality: Gaussian case). *Let A be a square matrix. If X and Y are two independent standard Gaussian vectors, we have:*

$$\mathbb{P}\left(|X'AX - \mathbb{E}[X'AX]| \geq u\right) \leq 2\exp\left(-c\min\left\{\frac{u^2}{\|A\|_F^2}, \frac{u}{\|A\|_2}\right\}\right)$$
$$\mathbb{P}\left(|X'AY - \mathbb{E}[X'AY]| \geq u\right) \leq 2\exp\left(-c\min\left\{\frac{u^2}{\|A\|_F^2}, \frac{u}{\|A\|_2}\right\}\right).$$

Proof. See Vershynin (2018, Theorem 6.2.1) for the first inequality. We will see that it implies the second one. Let us define

$$\widetilde{A} = \begin{bmatrix} 0 & A \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad \widetilde{X} = \begin{bmatrix} X \\ Y \end{bmatrix}.$$

We note that $\|\widetilde{A}\|_F = \|A\|_F$ and $\|\widetilde{A}\|_2 = \|A\|_2$. Applying the first inequality to $\widetilde{X}'\widetilde{A}\widetilde{X} = X'AY$ yields the expected result. □

A.3 Differentiation

Lemma A.3.1 (Danskin-Bertsekas). *Let $\varphi\colon \mathbb{R}^n \times \mathbb{R}^m \longrightarrow (-\infty, +\infty]$ be a function, and let $\mathcal{Y} \subset \mathbb{R}^m$ be a compact set such that $\varphi(\cdot, y)$ is closed proper convex for every $y \in \mathcal{Y}$. We consider the partial maximization*

$$f\colon x \in \mathbb{R}^n \longmapsto \max_{y \in \mathcal{Y}} \varphi(x,y)$$

If $\mathcal{X} = \operatorname{int}(\operatorname{dom}(f)) \neq \emptyset$ and φ is continuous on $\mathcal{X} \times \mathcal{Y}$, then for every $x \in \mathcal{X}$, we have

$$\partial f(x) = \operatorname{conv}\{\partial \varphi(x,\bar{y}) : \bar{y} \in \operatorname*{argmin}_{y \in \mathcal{Y}} \varphi(x,y)\}$$

Proof. See Danskin (1967) and Bertsekas (1971, Proposition A.22). □

Bibliography

Because that's what Hermione does.
When in doubt, go to the library.

Ron Weasley
Harry Potter and the
Chamber of Secrets (2002)

Agrawal, A., B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter (2019). "Differentiable Convex Optimization Layers". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. (cit. on p. 133).

Agrawal, A., R. Verschueren, S. Diamond, and S. Boyd (2018). "A Rewriting System for Convex Optimization Problems". In: *Journal of Control and Decision* 5.1, pp. 42–60. DOI: 10/ggkj28 (cit. on p. 207).

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). *Network Flows: Theory, Algorithms, and Applications*. Anglais. Englewood Cliffs, N.J: Pearson. 864 pp. ISBN: 978-0-13-617549-0 (cit. on p. 220).

Amos, B. and J. Z. Kolter (2017). "OptNet: Differentiable Optimization as a Layer in Neural Networks". en. In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 136–145 (cit. on pp. 133, 134).

Arthaud, F., G. Lecoeur, and A. Pierre (2021). *Transformers à Grande Vitesse*. URL: http://arxiv.org/abs/2105.08526. preprint (cit. on p. 202).

Barbour, W., J. C. Martinez Mori, S. Kuppa, and D. B. Work (2018). "Prediction of Arrival Times of Freight Traffic on US Railroads Using Support Vector Regression". In: *Transportation Research Part C: Emerging Technologies* 93, pp. 211–227. ISSN: 0968-090X. DOI: 10.1016/j.trc.2018.05.019 (cit. on p. 202).

Barto, A. G. and S. Mahadevan (2003). "Recent Advances in Hierarchical Reinforcement Learning". en. In: *Discrete Event Dynamic Systems* 13.1, pp. 41–77. ISSN: 1573-7594. DOI: 10.1023/A:1022140919877 (cit. on p. 135).

Bast, H., D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck (2016). "Route Planning in Transportation Networks". en. In: *Algorithm Engineering:*

Selected Results and Surveys. Ed. by L. Kliemann and P. Sanders. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 19–80. ISBN: 978-3-319-49487-6. DOI: 10.1007/978-3-319-49487-6_2 (cit. on p. 75).

Basu, S. and G. Michailidis (2015). "Regularized Estimation in Sparse High-Dimensional Time Series Models". In: *The Annals of Statistics* 43.4. ISSN: 0090-5364. DOI: 10.1214/15-AOS1315 (cit. on p. 88).

Baum, L. E., T. Petrie, G. Soules, and N. Weiss (1970). "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". In: *The Annals of Mathematical Statistics* 41.1, pp. 164–171. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177697196 (cit. on pp. 58, 59).

Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (2018). "Automatic Differentiation in Machine Learning: A Survey". In: *Journal of Machine Learning Research* 18.153, pp. 1–43. ISSN: 1533-7928 (cit. on pp. 45, 46, 133).

Ben-Tal, A. (1980). "Characterization of Pareto and Lexicographic Optimal Solutions". en. In: *Multiple Criteria Decision Making Theory and Application*. Ed. by G. Fandel and T. Gal. Lecture Notes in Economics and Mathematical Systems. Berlin, Heidelberg: Springer, pp. 1–11. ISBN: 978-3-642-48782-8. DOI: 10.1007/978-3-642-48782-8_1 (cit. on p. 162).

Ben-Tal, A. and S. Zlobec (1977). "Convex Programming and the Lexicographic Multicriteria Problem". In: *Mathematische Operationsforschung und Statistik. Series Optimization* 8.1, pp. 61–73. ISSN: 0323-3898. DOI: 10.1080/02331937708842406 (cit. on p. 162).

Bengio, Y. and P. Frasconi (1994). "An Input Output HMM Architecture". In: *Advances in Neural Information Processing Systems*. Vol. 7. MIT Press (cit. on pp. 57, 59).

Bengio, Y., A. Lodi, and A. Prouvost (2021). "Machine Learning for Combinatorial Optimization: A Methodological Tour d'horizon". en. In: *European Journal of Operational Research* 290.2, pp. 405–421. ISSN: 03772217. DOI: 10.1016/j.ejor.2020.07.063 (cit. on pp. 128, 131, 218).

Berthet, Q., M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach (2020). "Learning with Differentiable Perturbed Optimizers". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 9508–9519 (cit. on pp. 129, 134, 135, 138–142, 145, 151, 154).

Bertsekas, D. P. (1971). "Control of Uncertain Systems with a Set-Membership Description of the Uncertainty." eng. PhD thesis. Massachusetts Institute of Technology (cit. on p. 233).

Besançon, M., A. Carderera, and S. Pokutta (2022). "FrankWolfe.jl: A High-Performance and Flexible Toolbox for Frank–Wolfe Algorithms and Conditional Gradients". en. In: *INFORMS Journal on Computing*, ijoc.2022.1191. ISSN: 1091-9856, 1526-5528. DOI: 10.1287/ijoc.2022.1191 (cit. on p. 146).

Besançon, M., T. Papamarkou, D. Anthoff, A. Arslan, S. Byrne, D. Lin, and J. Pearson (2021). "Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem". en. In: *Journal of Statistical Software* 98, pp. 1–30. ISSN: 1548-7660. DOI: 10.18637/jss.v098.i16 (cit. on p. 62).

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). "Julia: A Fresh Approach to Numerical Computing". en. In: *SIAM Review* 59.1, pp. 65–98. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/141000671 (cit. on pp. 24, 33, 42, 95, 132).

Blondel, M., Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert (2022). "Efficient and Modular Implicit Differentiation". en. In: Advances in Neural Information Processing Systems (cit. on pp. 48, 49, 133, 145, 147).

Blondel, M., F. Llinares-López, R. Dadashi, L. Hussenot, and M. Geist (2022). "Learning Energy Networks with Generalized Fenchel-Young Losses". en. In: Advances in Neural Information Processing Systems (cit. on p. 224).

Blondel, M., A. F. T. Martins, and V. Niculae (2020). "Learning with Fenchel-Young Losses". In: *Journal of Machine Learning Research* 21.35, pp. 1–69. ISSN: 1533-7928 (cit. on pp. 132, 134, 138, 146, 151, 152, 154, 183).

Blondel, M., O. Teboul, Q. Berthet, and J. Djolonga (2020). "Fast Differentiable Sorting and Ranking". en. In: *Proceedings of the 37th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 950–959 (cit. on p. 134).

Bonnans, J.-F., J. C. Gilbert, C. Lemarechal, and C. A. Sagastizábal (2006). *Numerical Optimization: Theoretical and Practical Aspects*. en. Springer Science & Business Media. 492 pp. ISBN: 978-3-540-35447-5 (cit. on p. 179).

Bonnel, H., A. N. Iusem, and B. F. Svaiter (2005). "Proximal Methods in Vector Optimization". In: *SIAM Journal on Optimization* 15.4, pp. 953–970. ISSN: 1052-6234. DOI: 10.1137/S1052623403429093 (cit. on p. 162).

Borndörfer, R., T. Klug, L. Lamorgese, C. Mannino, M. Reuther, and T. Schlechte, eds. (2018). *Handbook of Optimization in the Railway Industry*. en. International Series in Operations Research & Management Science. Springer International Publishing. ISBN: 978-3-319-72152-1. DOI: 10.1007/978-3-319-72153-8 (cit. on pp. 17, 27).

Borwein, J. and A. S. Lewis (2010). *Convex Analysis and Nonlinear Optimization: Theory and Examples*. en. Springer Science & Business Media. 316 pp. ISBN: 978-0-387-31256-9 (cit. on p. 178).

Bot, R. I., S.-M. Grad, and G. Wanka (2009). *Duality in Vector Optimization*. en. Springer Science & Business Media. 408 pp. ISBN: 978-3-642-02886-1 (cit. on pp. 160, 162, 163, 165).

Boyarski, E., A. Felner, P. L. Bodic, D. D. Harabor, P. J. Stuckey, and S. Koenig (2021). "Further Improved Heuristics For Conflict-Based Search". en. In: *Proceedings of the International Symposium on Combinatorial Search* 12.1 (1), pp. 213–215. ISSN: 2832-9163. DOI: 10.1609/socs.v12i1.18587 (cit. on p. 74).

Boyarski, E., A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony (2015). "ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding". en. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. ISBN: 978-1-57735-738-4 (cit. on p. 74).

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. en. Cambridge University Press. 744 pp. ISBN: 978-0-521-83378-3 (cit. on p. 161).

Bubeck, S. (2015). "Convex Optimization: Algorithms and Complexity". English. In: *Foundations and Trends in Machine Learning* 8.3-4, pp. 231–357. ISSN: 1935-8237, 1935-8245. DOI: 10/gc7rf3 (cit. on p. 173).

Büker, T. and B. Seybold (2012). "Stochastic Modelling of Delay Propagation in Large Networks". In: *Journal of Rail Transport Planning & Management* 2.1, pp. 34–50. ISSN: 2210-9706. DOI: 10.1016/j.jrtpm.2012.10.001 (cit. on p. 202).

Buonaccorsi, J. P. (2010). *Measurement Error: Models, Methods, and Applications*. New York: Chapman and Hall/CRC. 464 pp. ISBN: 978-0-429-15035-7. DOI: 10.1201/9781420066586 (cit. on p. 88).

Burdett, R. and E. Kozan (2014). "Determining Operations Affected by Delay in Predictive Train Timetables". In: *Computers & Operations Research* 41, pp. 150–166. ISSN: 0305-0548. DOI: 10.1016/j.cor.2013.08.011 (cit. on p. 201).

Cacchiani, V., D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar (2014). "An Overview of Recovery Models and Algorithms for Real-Time Railway Rescheduling". In: *Transportation Research Part B: Methodological* 63, pp. 15–37. ISSN: 0191-2615. DOI: 10.1016/j.trb.2014.01.009 (cit. on p. 216).

Candes, E. and T. Tao (2007). "The Dantzig Selector: Statistical Estimation When p Is Much Larger than n". In: *The Annals of Statistics* 35.6. ISSN: 0090-5364. DOI: 10.1214/009053606000001523 (cit. on p. 89).

Cappé, O., E. Moulines, and T. Rydén (2005). *Inference in Hidden Markov Models*. en. Springer Series in Statistics. New York, NY: Springer New York. ISBN: 978-0-387-40264-2 978-0-387-28982-3. DOI: 10.1007/0-387-28982-8 (cit. on pp. 57, 88).

Chami, I., S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy (2022). "Machine Learning on Graphs: A Model and Comprehensive Taxonomy". In: *Journal of Machine Learning Research* 23.89, pp. 1–64. ISSN: 1533-7928 (cit. on p. 218).

Chandesris, M. and X. Chapuis (2018). "Non-Parametric Approach for Real Time Prediction". en. In: Conference on Advanced Systems in Public Transport and TransitData, p. 14 (cit. on p. 203).

Chen, M., R. A. Chowdhury, V. Ramachandran, D. L. Roche, and L. Tong (2007). *Priority Queues and Dijkstra's Algorithm*. en. UTCS Technical Report TR-07-54. University of Texas Austin, p. 25 (cit. on p. 77).

Chung, K.-M., H. Lam, Z. Liu, and M. Mitzenmacher (2012). "Chernoff-Hoeffding Bounds for Markov Chains: Generalized and Simplified". In: *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Ed. by C. Dürr and T. Wilke. Vol. 14. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 124–135. ISBN: 978-3-939897-35-4. DOI: 10.4230/LIPIcs.STACS.2012.124 (cit. on p. 232).

Corman, F. and P. Kecman (2018). "Stochastic Prediction of Train Delays in Real-Time Using Bayesian Networks". In: *Transportation Research Part C: Emerging Technologies* 95, pp. 599–615. ISSN: 0968-090X. DOI: 10.1016/j.trc.2018.08.003 (cit. on p. 202).

Cover, T. M. and J. A. Thomas (2006). *Elements of Information Theory*. 2nd ed. Hoboken, N.J: Wiley-Interscience. 748 pp. ISBN: 978-0-471-24195-9 (cit. on p. 231).

Cox, D. R. (1972). "Regression Models and Life-Tables". en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 34.2, pp. 187–202. ISSN: 2517-6161. DOI: 10.1111/j.2517-6161.1972.tb00899.x (cit. on p. 193).

D., G. (2022). "Recherche d'itinéraires dans un réseau ferroviaire : apprendre à mieux optimiser". French. Journées SMAI MODE 2022 (Limoges) (cit. on p. 213).

D., G., L. Baty, L. Bouvier, and A. Parmentier (2022). *Learning with Combinatorial Optimization Layers: A Probabilistic Approach*. DOI: 10.48550/arXiv.2207.13513. URL: http://arxiv.org/abs/2207.13513. preprint (cit. on p. 128).

D., G., L. Bouvier, and L. Baty (2022). "InferOpt.jl: Combinatorial Optimization in ML Pipelines". en. Conference talk. JuliaCon 2022 (cit. on p. 128).

D., G. and Y. de Castro (2022). *Minimax Estimation of Partially-Observed Vector AutoRegressions*. DOI: 10.48550/arXiv.2106.09327. URL: http://arxiv.org/abs/2106.09327. preprint (cit. on pp. 86, 199).

D., G. and A. Parmentier (2022). “Learning to Solve Stochastic Multi-Agent Path Finding”. English. 23ème Congrès Annuel de La Société Française de Recherche Opérationnelle et d’Aide à La Décision (Villeurbanne - Lyon, France) (cit. on p. 213).

D., G. and M. Tarek (2022). “ImplicitDifferentiation.jl: Differentiating Implicit Functions”. en. Conference talk. JuliaCon 2022 (cit. on p. 42).

Da Cruz Neto, J. X., G. J. P. Da Silva, O. P. Ferreira, and J. O. Lopes (2013). “A Subgradient Method for Multiobjective Optimization”. en. In: *Computational Optimization and Applications* 54.3, pp. 461–472. ISSN: 1573-2894. DOI: 10.1007/s10589-012-9494-7 (cit. on p. 162).

Daley, D. J. and D. Vere-Jones (2003). *An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods*. en. 2nd ed. Probability and Its Applications, An Introduction to the Theory of Point Processes. New York: Springer-Verlag. ISBN: 978-0-387-95541-4. DOI: 10.1007/b97277 (cit. on p. 54).

Danskin, J. M. (1967). *The Theory of Max-Min and Its Application to Weapons Allocation Problems*. Red. by M. Beckmann, R. Henn, A. Jaeger, W. Krelle, H. P. Künzi, K. Wenke, and Ph. Wolfe. Vol. 5. Ökonometrie Und Unternehmensforschung / Econometrics and Operations Research. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-46094-4 978-3-642-46092-0. DOI: 10.1007/978-3-642-46092-0 (cit. on p. 233).

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22. ISSN: 2517-6161. DOI: 10.1111/j.2517-6161.1977.tb01600.x (cit. on p. 59).

Diamond, S. and S. Boyd (2016). “CVXPY: A Python-Embedded Modeling Language for Convex Optimization”. In: *Journal of Machine Learning Research* 17.83, pp. 1–5. ISSN: 1533-7928 (cit. on p. 207).

Dib, A. (2021). “High Dimensional Pattern Learning Applied to Symbolic Time-Series”. PhD thesis. Université Paris-Saclay (cit. on p. 194).

Dinitz, M., S. Im, T. Lavastida, B. Moseley, and S. Vassilvitskii (2021). “Faster Matchings via Learned Duals”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 10393–10406 (cit. on p. 220).

Domahidi, A., E. Chu, and S. Boyd (2013). “ECOS: An SOCP Solver for Embedded Systems”. In: 2013 European Control Conference (ECC), pp. 3071–3076. DOI: 10.23919/ECC.2013.6669541 (cit. on p. 207).

Douc, R., E. Moulines, and D. Stoffer (2013). *Nonlinear Time Series: Theory, Methods and Applications with R Examples*. New York: Chapman and Hall/CRC. 551 pp. ISBN: 978-0-429-11263-8. DOI: 10.1201/b16331 (cit. on p. 93).

Doucet, A., S. Godsill, and C. Andrieu (2000). “On Sequential Monte Carlo Sampling Methods for Bayesian Filtering”. en. In: *Statistics and Computing* 10.3, pp. 197–208. ISSN: 1573-1375. DOI: 10.1023/A:1008935410038 (cit. on p. 88).

Drummond, L. M. G. and B. F. Svaiter (2005). “A Steepest Descent Method for Vector Optimization”. en. In: *Journal of Computational and Applied Mathematics* 175.2, pp. 395–414. ISSN: 0377-0427. DOI: 10.1016/j.cam.2004.06.018 (cit. on p. 162).

Drummond, L. G. and A. Iusem (2004). “A Projected Gradient Method for Vector Optimization Problems”. en. In: *Computational Optimization and Applications* 28.1, pp. 5–29. ISSN: 1573-2894. DOI: 10.1023/B:COAP.0000018877.86161.8b (cit. on p. 162).

Dubhashi, D. P. and A. Panconesi (2009). *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge: Cambridge University Press. ISBN: 978-0-521-88427-3. DOI: 10.1017/CBO9780511581274 (cit. on p. 232).

Duchi, J. (2007). "Derivations for Linear Algebra and Optimization" (cit. on p. 231).

— (2019). "Information Theory and Statistics". Course notes for Statistics 311/Electrical Engineering 377. Stanford University (cit. on p. 116).

Dunning, I., J. Huchette, and M. Lubin (2017). "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2, pp. 295–320. ISSN: 0036-1445. DOI: 10/gftshn (cit. on p. 95).

Ehrgott, M. (2005). *Multicriteria Optimization*. en. Springer Science & Business Media. 329 pp. ISBN: 978-3-540-21398-7 (cit. on pp. 160, 162).

Eichfelder, G. (2021). "Twenty Years of Continuous Multiobjective Optimization in the Twenty-First Century". en. In: *EURO Journal on Computational Optimization* 9, p. 100014. ISSN: 2192-4406. DOI: 10.1016/j.ejco.2021.100014 (cit. on p. 161).

Eisner, J. (2016). "Inside-Outside and Forward-Backward Algorithms Are Just Backprop (Tutorial Paper)". In: *Proceedings of the Workshop on Structured Prediction for NLP*. Austin, TX: Association for Computational Linguistics, pp. 1–17. DOI: 10.18653/v1/W16-5901 (cit. on p. 59).

Elmachtoub, A. N. and P. Grigas (2022). "Smart "Predict, Then Optimize"". en. In: *Management Science* 68.1, pp. 9–26. ISSN: 0025-1909, 1526-5501. DOI: 10.1287/mnsc.2020.3922 (cit. on pp. 131, 134, 151, 153).

Fairbanks, J., M. Besançon, S. Simon, J. Hoffiman, N. Eubank, and S. Karpinski (2021). *JuliaGraphs/Graphs.jl: An Optimized Graphs Package for the Julia Programming Language* (cit. on pp. 76, 136, 224).

Felner, A., R. Stern, S. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek (2017). "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges". en. In: *Proceedings of the International Symposium on Combinatorial Search* 8.1 (1), pp. 29–37. ISSN: 2832-9163. DOI: 10.1609/socs.v8i1.18423 (cit. on p. 73).

Ferber, A., B. Wilder, B. Dilkina, and M. Tambe (2020). "MIPaaL: Mixed Integer Program as a Layer". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.02, pp. 1504–1511. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i02.5509 (cit. on p. 134).

Fliege, J., L. M. G. Drummond, and B. F. Svaiter (2009). "Newton's Method for Multiobjective Optimization". In: *SIAM Journal on Optimization* 20.2, pp. 602–626. ISSN: 1052-6234. DOI: 10.1137/08071692X (cit. on p. 162).

Fliege, J. and B. F. Svaiter (2000). "Steepest Descent Methods for Multicriteria Optimization". en. In: *Mathematical Methods of Operations Research* 51.3, pp. 479–494. ISSN: 1432-5217. DOI: 10.1007/s001860000043 (cit. on p. 162).

Flier, H., R. Gelashvili, T. Graffagnino, and M. Nunkesser (2009). "Mining Railway Delay Dependencies in Large-Scale Real-World Delay Data". en. In: *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*. Ed. by R. K. Ahuja, R. H. Möhring, and C. D. Zaroliagis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 354–368. ISBN: 978-3-642-05465-5. DOI: 10.1007/978-3-642-05465-5_15 (cit. on p. 202).

Frank, M. and P. Wolfe (1956). "An Algorithm for Quadratic Programming". en. In: *Naval Research Logistics Quarterly* 3.1-2, pp. 95–110. ISSN: 00281441, 19319193. DOI: 10.1002/nav.3800030109 (cit. on p. 145).

Ge, H., K. Xu, and Z. Ghahramani (2018). "Turing: A Language for Flexible Probabilistic Inference". en. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, pp. 1682–1690 (cit. on p. 224).

Gebken, B. and S. Peitz (2021). "An Efficient Descent Method for Locally Lipschitz Multiobjective Optimization Problems". en. In: *Journal of Optimization Theory and Applications* 188.3, pp. 696–723. ISSN: 1573-2878. DOI: 10.1007/s10957-020-01803-w (cit. on p. 162).

Geoffrion, A. M. (1968). "Proper Efficiency and the Theory of Vector Maximization". en. In: *Journal of Mathematical Analysis and Applications* 22.3, pp. 618–630. ISSN: 0022-247X. DOI: 10.1016/0022-247X(68)90201-1 (cit. on p. 161).

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. eng. Cambridge, Massachusetts: The MIT Press. ISBN: 978-0-262-33737-3 (cit. on pp. 22, 32, 133).

Goverde, R. M. P. (2007). "Railway Timetable Stability Analysis Using Max-plus System Theory". In: *Transportation Research Part B: Methodological*. Advanced Modelling of Train Operations in Stations and Networks 41.2, pp. 179–201. ISSN: 0191-2615. DOI: 10.1016/j.trb.2006.02.003 (cit. on p. 201).

Griewank, A. and A. Walther (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics. 438 pp. ISBN: 978-0-89871-659-7 (cit. on pp. 45, 46, 133).

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). "Exploring Network Structure, Dynamics, and Function Using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, pp. 11–15 (cit. on pp. 207, 224).

Han, F., H. Lu, and H. Liu (2015). "A Direct Estimation of High Dimensional Stationary Vector Autoregressions". In: *Journal of Machine Learning Research* 16.97, pp. 3115–3150. ISSN: 1533-7928 (cit. on pp. 89, 90, 98, 112, 113).

Hart, P. E., N. J. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. ISSN: 2168-2887. DOI: 10/c9zgc4 (cit. on p. 75).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (cit. on p. 136).

Horn, R. A. and C. R. Johnson (1991). *Topics in Matrix Analysis*. Cambridge: Cambridge University Press. ISBN: 978-0-521-46713-1. DOI: 10.1017/CBO9780511840371 (cit. on p. 229).

— (2012). *Matrix Analysis*. 2nd ed. Cambridge: Cambridge University Press. ISBN: 978-0-521-83940-2 978-1-139-02041-1 978-0-521-54823-6. DOI: 10.1017/CBO9781139020411 (cit. on p. 229).

Huangfu, Q. and J. A. J. Hall (2018). "Parallelizing the Dual Revised Simplex Method". en. In: *Mathematical Programming Computation* 10.1, pp. 119–142. ISSN: 1867-2957. DOI: 10.1007/s12532-017-0130-5 (cit. on p. 95).

Innes, M. (2019). *Don't Unroll Adjoint: Differentiating SSA-Form Programs*. URL: http://arxiv.org/abs/1810.07951. preprint (cit. on pp. 47, 136).

Innes, M., E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah (2018). *Fashionable Modelling with Flux*. URL: http://arxiv.org/abs/1811.01457. preprint (cit. on p. 136).

Innes, M. (2018). "Flux: Elegant Machine Learning with Julia". In: *Journal of Open Source Software* 3.25, p. 602. ISSN: 2475-9066. DOI: 10.21105/joss.00602 (cit. on pp. 48, 136).

Isermann, H. (1982). "Linear Lexicographic Optimization". en. In: *Operations-Research-Spektrum* 4.4, pp. 223–228. ISSN: 1436-6304. DOI: 10/fvqk7b (cit. on p. 162).

Jaggi, M. (2013). "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization". en. In: *Proceedings of the 30th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 427–435 (cit. on p. 145).

Jahn, J. (2010). *Vector Optimization: Theory, Applications, and Extensions*. en. Springer Science & Business Media. 490 pp. ISBN: 978-3-642-17005-8 (cit. on pp. 160, 162).

Jalali, A. and R. Willett (2018). *Missing Data in Sparse Transition Matrix Estimation for Sub-Gaussian Vector Autoregressive Processes*. DOI: 10.48550/arXiv.1802.09511. URL: http://arxiv.org/abs/1802.09511. preprint (cit. on p. 88).

Jardine, A. K. S., D. Lin, and D. Banjevic (2006). "A Review on Machinery Diagnostics and Prognostics Implementing Condition-Based Maintenance". In: *Mechanical Systems and Signal Processing* 20.7, pp. 1483–1510. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2005.09.012 (cit. on p. 192).

Jones, D. R., C. D. Perttunen, and B. E. Stuckman (1993). "Lipschitzian Optimization without the Lipschitz Constant". en. In: *Journal of Optimization Theory and Applications* 79.1, pp. 157–181. ISSN: 0022-3239, 1573-2878. DOI: 10.1007/BF00941892 (cit. on p. 148).

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems". en. In: *Journal of Basic Engineering* 82.1, pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552 (cit. on pp. 88, 96).

Kecman, P. and R. M. P. Goverde (2015). "Online Data-Driven Adaptive Prediction of Train Event Times". In: *IEEE Transactions on Intelligent Transportation Systems* 16.1, pp. 465–474. ISSN: 1524-9050. DOI: 10.1109/TITS.2014.2347136 (cit. on p. 201).

Kecman, P., F. Corman, and L. Meng (2015). "Train Delay Evolution as a Stochastic Process". en. In: 6th International Conference on Railway Operations Modelling and Analysis (RailTokyo2015). IVT, ETH Zurich; Orange Labs. DOI: 10.3929/ethz-b-000183322 (cit. on p. 202).

Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: 3rd International Conference on Learning Representations. Ed. by Y. Bengio and Y. LeCun. San Diego, California, United States (cit. on p. 155).

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). "Optimization by Simulated Annealing". In: *Science* 220.4598, pp. 671–680. DOI: 10.1126/science.220.4598.671 (cit. on p. 80).

Klein, J. P., H. C. van Houwelingen, J. G. Ibrahim, and T. H. Scheike (2013). *Handbook of Survival Analysis*. CRC Press. ISBN: 1-4665-5566-1 (cit. on p. 193).

Kochenderfer, M. J. and T. A. Wheeler (2019). *Algorithms for Optimization*. Anglais. Cambridge, Massachusetts: The MIT Press. 520 pp. ISBN: 978-0-262-03942-0 (cit. on p. 48).

Kock, A. B. and L. Callot (2015). "Oracle Inequalities for High Dimensional Vector Autoregressions". en. In: *Journal of Econometrics*. High Dimensional Problems in Econometrics 186.2, pp. 325–344. ISSN: 0304-4076. DOI: 10.1016/j.jeconom.2015.02.013 (cit. on p. 88).

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. en. MIT Press. 1268 pp. ISBN: 978-0-262-01319-2 (cit. on p. 202).

Kolter, J. Z., D. Duvenaud, and M. Johnson (2020). *Deep Implicit Layers - Neural ODEs, Deep Equilibirum Models, and Beyond*. en. URL: http://implicit-layers-tutorial.org/ (cit. on pp. 49, 133).

Korte, B. and J. Vygen (2006). *Combinatorial Optimization: Theory and Algorithms*. eng. 3rd edition. Algorithms and Combinatorics 21. Berlin: Springer. ISBN: 978-3-540-29297-5 (cit. on pp. 17, 27, 130).

Kotary, J., F. Fioretto, P. V. Hentenryck, and B. Wilder (2021). "End-to-End Constrained Optimization Learning: A Survey". en. In: Twenty-Ninth International Joint Conference on Artificial Intelligence. Vol. 5, pp. 4475–4482. DOI: 10.24963/ijcai.2021/610 (cit. on p. 128).

Lair, W. (2011). "Modélisation Dynamique de Systèmes Complexes Pour Le Calcul de Grandeurs Fiabilistes et l'optimisation de La Maintenance". PhD thesis. Université de Pau et des Pays de l'Adour (cit. on p. 194).

Laurent, F., M. Schneider, C. Scheller, J. Watson, J. Li, Z. Chen, Y. Zheng, S.-H. Chan, K. Makhnev, O. Svidchenko, V. Egorov, D. Ivanov, A. Shpilman, E. Spirovska, O. Tanevski, A. Nikov, R. Grunder, D. Galevski, J. Mitrovski, G. Sartoretti, Z. Luo, M. Damani, N. Bhattacharya, S. Agarwal, A. Egli, E. Nygren, and S. Mohanty (2021). *Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World*. URL: http://arxiv.org/abs/2103.16511. preprint (cit. on p. 216).

Levin, D. A., Y. Peres, E. L. Wilmer, J. Propp, and D. B. Wilson (2017). *Markov Chains and Mixing Times*. Second edition. Providence, Rhode Island: American Mathematical Society. 447 pp. ISBN: 978-1-4704-2962-1 (cit. on p. 232).

Li, J., Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig (2021). "Anytime Multi-Agent Path Finding via Large Neighborhood Search". en. In: Twenty-Ninth International Joint Conference on Artificial Intelligence. Vol. 4, pp. 4127–4135. DOI: 10.24963/ijcai.2021/568 (cit. on pp. 74, 78, 79).

— (2022). "MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.9 (9), pp. 10256–10265. ISSN: 2374-3468. DOI: 10.1609/aaai.v36i9.21266 (cit. on pp. 74, 79).

Li, J., Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig (2021). "Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge". en. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 31, pp. 477–485. ISSN: 2334-0843 (cit. on p. 216).

Lin, D., S. Byrne, J. M. White, D. Widmann, A. Noack, M. Besançon, D. Bates, J. Pearson, J. Zito, A. Arslan, M. Schauer, K. Squire, D. Anthoff, T. Papamarkou, J. Drugowitsch, B. Deonovic, A. Sengupta, G. Ragusa, G. Moynihan, B. J. Smith, M. O'Leary, Michael, M. Tarek, M. J. Innes, C. Dann, G. Lacerda, I. Dunning, J. Weidner, and J. Chen (2023). *JuliaStats/Distributions.Jl: V0.25.86*. Zenodo. DOI: 10.5281/zenodo.7695673 (cit. on p. 62).

Little, R. J. A. and D. B. Rubin (2020). *Statistical Analysis with Missing Data*. 3rd ed. Wiley Series in Probability and Statistics. Hoboken, NJ: Wiley. 1 p. ISBN: 978-1-118-59601-2 978-0-470-52679-8 (cit. on p. 88).

Loh, P.-L. and M. J. Wainwright (2012). "High-Dimensional Regression with Noisy and Missing Data: Provable Guarantees with Nonconvexity". In: *The Annals of Statistics* 40.3, pp. 1637–1664. ISSN: 0090-5364, 2168-8966. DOI: 10/ggx5bj (cit. on p. 88).

Luc, D. T. (2016). *Multiobjective Linear Programming*. en. Cham: Springer International Publishing. ISBN: 978-3-319-21090-2 978-3-319-21091-9. DOI: 10.1007/978-3-319-21091-9 (cit. on p. 162).

Lütkepohl, H. (2005). *New Introduction to Multiple Time Series Analysis*. en. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-40172-8 978-3-540-27752-1. DOI: 10.1007/978-3-540-27752-1 (cit. on pp. 88, 95).

Ma, H., D. Harabor, P. J. Stuckey, J. Li, and S. Koenig (2019). "Searching with Consistent Prioritization for Multi-Agent Path Finding". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (01), pp. 7643–7650. ISSN: 2374-3468. DOI: 10/ghkkbz (cit. on p. 78).

Malagò, L. and G. Pistone (2015). "Information Geometry of the Gaussian Distribution in View of Stochastic Optimization". en. In: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. FOGA'15: Foundations of Genetic Algorithms XIII. Aberystwyth United Kingdom: ACM, pp. 150–162. ISBN: 978-1-4503-3434-1. DOI: 10.1145/2725494.2725510 (cit. on p. 94).

Mandi, J., E. Demirović, P. J. Stuckey, and T. Guns (2020). "Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.02, pp. 1603–1610. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i02.5521 (cit. on p. 134).

Mandi, J. and T. Guns (2020). "Interior Point Solving for LP-based Prediction+optimisation". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 7272–7282 (cit. on p. 134).

Mann, T. P. (2006). "Numerically Stable Hidden Markov Model Implementation" (cit. on p. 60).

Margossian, C. C. (2019). "A Review of Automatic Differentiation and Its Efficient Implementation". en. In: *WIREs Data Mining and Knowledge Discovery* 9.4, e1305. ISSN: 1942-4795. DOI: 10.1002/widm.1305 (cit. on pp. 45, 46).

Martinez-Legaz, J. E. (1988). "Lexicographical Order and Duality in Multiobjective Programming". en. In: *European Journal of Operational Research* 33.3, pp. 342–348. ISSN: 0377-2217. DOI: 10.1016/0377-2217(88)90178-6 (cit. on p. 162).

Martins, A. F. T. and R. Astudillo (2016). "From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification". en. In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1614–1623 (cit. on p. 145).

Meester, L. E. and S. Muns (2007). "Stochastic Delay Propagation in Railway Networks and Phase-Type Distributions". In: *Transportation Research Part B: Methodological*. Advanced Modelling of Train Operations in Stations and Networks 41.2, pp. 218–230. ISSN: 0191-2615. DOI: 10.1016/j.trb.2006.02.007 (cit. on p. 202).

Melnyk, I. and A. Banerjee (2016). "Estimating Structured Vector Autoregressive Models". en. In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 830–839 (cit. on p. 88).

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. en. Springer Science & Business Media. 324 pp. ISBN: 978-0-7923-8278-2 (cit. on p. 160).

Mohanty, S., E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler (2020). *Flatland-RL : Multi-Agent Reinforcement Learning on Trains*. URL: http://arxiv.org/abs/2012.05893. preprint (cit. on pp. 18, 28, 214).

Monechi, B., P. Gravino, R. Di Clemente, and V. D. P. Servedio (2018). "Complex Delay Dynamics on Railway Networks from Universal Laws to Realistic Modelling". en. In: *EPJ Data Science* 7.1, p. 35. ISSN: 2193-1127. DOI: 10.1140/epjds/s13688-018-0160-x (cit. on p. 203).

Moses, W. and V. Churavy (2020). "Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 12472–12485 (cit. on p. 47).

Moses, W. S., V. Churavy, L. Paehler, J. Hückelheim, S. H. K. Narayanan, M. Schanen, and J. Doerfert (2021). "Reverse-Mode Automatic Differentiation and Optimization of GPU Kernels via Enzyme". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '21. New York, NY, USA: Association for Computing Machinery, pp. 1–16. ISBN: 978-1-4503-8442-1. DOI: 10.1145/3458817.3476165 (cit. on p. 47).

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. Cambridge, MA: MIT Press. 1067 pp. ISBN: 978-0-262-01802-9 (cit. on p. 27).

Niculae, V., A. F. T. Martins, M. Blondel, and C. Cardie (2018). "SparseMAP: Differentiable Sparse Structured Inference". en. In: *Proceedings of the 35th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 3799–3808 (cit. on p. 146).

Ogata, Y. (1981). "On Lewis' Simulation Method for Point Processes". In: *IEEE Transactions on Information Theory* 27.1, pp. 23–31. ISSN: 1557-9654. DOI: 10.1109/TIT.1981.1056305 (cit. on p. 55).

Oneto, L., E. Fumeo, G. Clerico, R. Canepa, F. Papa, C. Dambra, N. Mazzino, and D. Anguita (2018). "Train Delay Prediction Systems: A Big Data Analytics Perspective". In: *Big Data Research*. Selected Papers from the 2nd INNS Conference on Big Data: Big Data & Neural Networks 11, pp. 54–64. ISSN: 2214-5796. DOI: 10.1016/j.bdr.2017.05.002 (cit. on p. 202).

Orban, D. (2019). *Krylov.jl*. Zenodo. DOI: 10.5281/zenodo.6916375 (cit. on p. 50).

Orban, D. and A. S. Siqueira (2019). *LinearOperators.jl*. Zenodo. DOI: 10.5281/zenodo.6569329 (cit. on p. 50).

Pal, A. (2022). *Lux: Explicit Parameterization of Deep Neural Networks in Julia* (cit. on p. 66).

Parmentier, A. (2021a). *Learning Structured Approximations of Operations Research Problems*. URL: http://arxiv.org/abs/2107.04323. preprint (cit. on pp. 132, 148, 218).

— (2021b). "Learning to Approximate Industrial Problems by Operations Research Classic Problems". en. In: *Operations Research* 70.1, pp. 606–623. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.2020.2094 (cit. on pp. 134, 220, 221).

Paulus, A., M. Rolinek, V. Musil, B. Amos, and G. Martius (2021). "CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints". en. In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 8443–8453 (cit. on p. 134).

Peng, Y., M. Dong, and M. J. Zuo (2010). "Current Status of Machine Prognostics in Condition-Based Maintenance: A Review". en. In: *The International Journal of Advanced Manufacturing Technology* 50.1, pp. 297–313. ISSN: 1433-3015. DOI: 10.1007/s00170-009-2482-0 (cit. on p. 193).

Pereira, R., M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva (2017). "Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?" In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language*

Engineering. SLE 2017. New York, NY, USA: Association for Computing Machinery, pp. 256–267. ISBN: 978-1-4503-5525-4. DOI: 10.1145/3136014.3136031 (cit. on p. 42).

Peyré, G. and M. Cuturi (2019). "Computational Optimal Transport: With Applications to Data Science". English. In: *Foundations and Trends in Machine Learning* 11.5-6, pp. 355–607. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000073 (cit. on p. 50).

Phillips, M. and M. Likhachev (2011). "SIPP: Safe Interval Path Planning for Dynamic Environments". In: *2011 IEEE International Conference on Robotics and Automation*. 2011 IEEE International Conference on Robotics and Automation, pp. 5628–5635. DOI: 10/dz35tb (cit. on p. 80).

Pilanci, M. and S. Boyd (2021–2022). *EE364b - Convex Optimization II*. URL: https://stanford.edu/class/ee364b/index.html (cit. on p. 179).

Pineda, L., T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam (2022). *Theseus: A Library for Differentiable Nonlinear Optimization*. URL: http://arxiv.org/abs/2207.09442. preprint (cit. on p. 133).

Qin, F., A. Auerbach, and F. Sachs (2000). "A Direct Optimization Approach to Hidden Markov Modeling for Single Channel Kinetics". en. In: *Biophysical Journal* 79.4, pp. 1915–1927. ISSN: 0006-3495. DOI: 10.1016/S0006-3495(00)76441-1 (cit. on p. 59).

Rabiner, L. (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286. ISSN: 1558-2256. DOI: 10/cswph2 (cit. on pp. 57, 60).

Rao, M., T. Javidi, Y. C. Eldar, and A. Goldsmith (2017a). "Estimation in Autoregressive Processes with Partial Observations". In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4212–4216. DOI: 10.1109/ICASSP.2017.7952950 (cit. on pp. 89, 90, 109).

— (2017b). "Fundamental Estimation Limits in Autoregressive Processes with Compressive Measurements". In: 2017 IEEE International Symposium on Information Theory (ISIT), pp. 2895–2899. DOI: 10.1109/ISIT.2017.8007059 (cit. on pp. 89, 109).

Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. 248 pp. ISBN: 978-0-262-18253-9 (cit. on p. 224).

Rasmussen, J. G. (2018). *Lecture Notes: Temporal Point Processes and the Conditional Intensity Function*. URL: http://arxiv.org/abs/1806.00221. preprint (cit. on p. 54).

Reback, J., W. McKinney, jbrockmendel, J. V. den Bossche, T. Augspurger, P. Cloud, S. Hawkins, gfyoung, Sinhrks, M. Roeschke, A. Klein, T. Petersen, J. Tratner, C. She, W. Ayd, S. Naveh, patrick, M. Garcia, J. Schendel, A. Hayden, D. Saxton, V. Jancauskas, M. Gorelli, R. Shadrach, A. McMaster, P. Battiston, S. Seabold, K. Dong, chris-b1, and h-vetinari (2021). *Pandas-Dev/Pandas: Pandas 1.2.4*. Zenodo. DOI: 10.5281/zenodo.4681666 (cit. on p. 207).

Rentmeesters, M., W. Tsai, and K.-J. Lin (1996). "A Theory of Lexicographic Multi-Criteria Optimization". In: ICECCS '96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (Held Jointly with 6th CSESAW and 4th IEEE RTAW), pp. 76–79. DOI: 10.1109/ICECCS.1996.558386 (cit. on p. 162).

Revels, J., M. Lubin, and T. Papamarkou (2016). *Forward-Mode Automatic Differentiation in Julia*. DOI: 10.48550/arXiv.1607.07892. URL: http://arxiv.org/abs/1607.07892. preprint (cit. on p. 47).

Saad, Y. and M. H. Schultz (1986). "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3, pp. 856–869. ISSN: 0196-5204. DOI: 10.1137/0907058 (cit. on p. 50).

Salfner, F., M. Lenk, and M. Malek (2010). "A Survey of Online Failure Prediction Methods". In: *ACM Comput. Surv.* 42.3, 10:1–10:42. ISSN: 0360-0300. DOI: 10.1145/1670679.1670680 (cit. on p. 193).

Sammouri, W. (2014). "Data Mining of Temporal Sequences for the Prediction of Infrequent Failure Events : Application on Floating Train Data for Predictive Maintenance". PhD thesis. Université Paris Est (cit. on p. 194).

Schäfer, F., M. Tarek, L. White, and C. Rackauckas (2021). *AbstractDifferentiation.jl: Backend-Agnostic Differentiable Programming in Julia*. URL: http://arxiv.org/abs/2109.12449. preprint (cit. on p. 47).

Scherrer, C. and M. Schauer (2022). "Applied Measure Theory for Probabilistic Modeling". en. In: *Proceedings of the JuliaCon Conferences* 1.1, p. 92. ISSN: 2642-4029. DOI: 10.21105/jcon.00092 (cit. on p. 224).

Schlechte, T. (2012). "Railway Track Allocation: Models and Algorithms". en. PhD thesis. Technische Universität Berlin (cit. on pp. 16, 17, 26, 27).

Sen, P. K. (1968). "Estimates of the Regression Coefficient Based on Kendall's Tau". In: *Journal of the American Statistical Association* 63.324, pp. 1379–1389. ISSN: 0162-1459. DOI: 10/gfxz87 (cit. on p. 96).

Sharma, A., M. Besançon, J. D. Garcia, and B. Legat (2022). *Flexible Differentiable Optimization via Model Transformations*. URL: http://arxiv.org/abs/2206.06135. preprint (cit. on p. 133).

Sharon, G., R. Stern, A. Felner, and N. Sturtevant (2012). "Conflict-Based Search For Optimal Multi-Agent Path Finding". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1 (1), pp. 563–569. ISSN: 2374-3468. DOI: 10.1609/aaai.v26i1.8140 (cit. on p. 74).

Sharon, G., R. Stern, M. Goldenberg, and A. Felner (2011). "The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding". en. In: Twenty-Second International Joint Conference on Artificial Intelligence (cit. on p. 73).

Shelton, C. R. and G. Ciardo (2014). "Tutorial on Structured Continuous-Time Markov Processes". en. In: *Journal of Artificial Intelligence Research* 51, pp. 725–778. ISSN: 1076-9757. DOI: 10.1613/jair.4415 (cit. on p. 197).

Sherali, H. D. and A. L. Soyster (1983). "Preemptive and Nonpreemptive Multi-Objective Programming: Relationship and Counterexamples". en. In: *Journal of Optimization Theory and Applications* 39.2, pp. 173–186. ISSN: 1573-2878. DOI: 10.1007/BF00934527 (cit. on p. 162).

Shumway, R. H. and D. S. Stoffer (1982). "An Approach to Time Series Smoothing and Forecasting Using the EM Algorithm". en. In: *Journal of Time Series Analysis* 3.4, pp. 253–264. ISSN: 1467-9892. DOI: 10.1111/j.1467-9892.1982.tb00349.x (cit. on p. 88).

Si, X.-S., W. Wang, C.-H. Hu, and D.-H. Zhou (2011). "Remaining Useful Life Estimation – A Review on the Statistical Data Driven Approaches". In: *European Journal of Operational Research* 213.1, pp. 1–14. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2010.11.018 (cit. on p. 193).

Silver, D. (2005). "Cooperative Pathfinding". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 1.1 (1), pp. 117–122. ISSN: 2334-0924. DOI: 10.1609/aiide.v1i1.18726 (cit. on pp. 74, 78).

Simchowitz, M., H. Mania, S. Tu, M. I. Jordan, and B. Recht (2018). "Learning Without Mixing: Towards A Sharp Analysis of Linear System Identification". en. In: *Proceedings of the 31st Conference On Learning Theory*. Conference On Learning Theory. PMLR, pp. 439–473 (cit. on p. 93).

Spanninger, T., A. Trivella, B. Büchel, and F. Corman (2022). "A Review of Train Delay Prediction Approaches". en. In: *Journal of Rail Transport Planning & Management* 22, p. 100312. ISSN: 2210-9706. DOI: 10.1016/j.jrtpm.2022.100312 (cit. on p. 201).

Standley, T. (2010). "Finding Optimal Solutions to Cooperative Pathfinding Problems". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1 (1), pp. 173–178. ISSN: 2374-3468. DOI: 10.1609/aaai.v24i1.7564 (cit. on p. 73).

Stern, R. (2019). "Multi-Agent Path Finding – An Overview". en. In: *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*. Ed. by G. S. Osipov, A. I. Panov, and K. S. Yakovlev. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 96–115. ISBN: 978-3-030-33274-7. DOI: 10.1007/978-3-030-33274-7_6 (cit. on p. 73).

Stern, R., N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, R. Barták, and E. Boyarski (2019). "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". en. In: *Proceedings of the International Symposium on Combinatorial Search* 10.1 (1), pp. 151–158. ISSN: 2832-9163. DOI: 10.1609/socs.v10i1.18510 (cit. on pp. 70, 71, 80).

Sturtevant, N. R. (2012). "Benchmarks for Grid-Based Pathfinding". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2, pp. 144–148. ISSN: 1943-0698. DOI: 10.1109/TCIAIG.2012.2197681 (cit. on p. 80).

Surynek, P. (2010). "An Optimization Variant of Multi-Robot Path Planning Is Intractable". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1 (1), pp. 1261–1263. ISSN: 2374-3468 (cit. on p. 73).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press. 526 pp. ISBN: 978-0-262-03924-6 (cit. on p. 135).

Tang, B. and E. B. Khalil (2022). *PyEPO: A PyTorch-based End-to-End Predict-then-Optimize Library for Linear and Integer Programming*. URL: http://arxiv.org/abs/2206.14234. preprint (cit. on p. 134).

Tellache, N. E. H., F. Meunier, and A. Parmentier (2022). *Linear Lexicographic Optimization and Preferential Bidding System*. URL: http://arxiv.org/abs/2201.08907. preprint (cit. on p. 162).

Tibshirani, R. (1996). "Regression Shrinkage and Selection Via the Lasso". en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1, pp. 267–288. ISSN: 00359246. DOI: 10.1111/j.2517-6161.1996.tb02080.x (cit. on p. 88).

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun (2005). "Large Margin Methods for Structured and Interdependent Output Variables". In: *Journal of Machine Learning Research* 6.50, pp. 1453–1484. ISSN: 1533-7928 (cit. on pp. 134, 152).

Tsybakov, A. B. (2009). *Introduction to Nonparametric Estimation*. en. Springer Series in Statistics. New York, NY: Springer New York. ISBN: 978-0-387-79051-0 978-0-387-79052-7. DOI: 10.1007/b13794 (cit. on pp. 116, 231, 233).

Udell, M. (2017). *ORIE 6326: Convex Optimization*. URL: https://people.orie.cornell.edu/mru8/orie6326/syllabus.html (cit. on p. 177).

Vershynin, R. (2018). *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge: Cambridge University Press. ISBN: 978-1-108-41519-4. DOI: 10.1017/9781108231596 (cit. on p. 233).

Vlastelica, M., A. Paulus, V. Musil, G. Martius, and M. Rolinek (2020). "Differentiation of Blackbox Combinatorial Solvers". en. In: 8th International Conference on Learning Representations (cit. on pp. 129, 134–136, 139, 155).

Wainwright, M. J. (2019). *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge: Cambridge University Press. ISBN: 978-1-108-49802-9. DOI: 10.1017/9781108627771 (cit. on p. 116).

Wei, L. J. (1992). "The Accelerated Failure Time Model: A Useful Alternative to the Cox Regression Model in Survival Analysis". en. In: *Statistics in Medicine* 11.14-15, pp. 1871–1879. ISSN: 1097-0258. DOI: 10.1002/sim.4780111409 (cit. on p. 193).

Wes McKinney (2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and Jarrod Millman. Vol. 445, pp. 56–61. DOI: 10/ggr6q3 (cit. on p. 207).

White, F. C., M. Abbott, M. Zgubic, J. Revels, S. Axen, A. Arslan, S. Schaub, N. Robinson, Y. Ma, G. Dhingra, W. Tebbutt, N. Heim, D. Widmann, A. D. W. Rosemberg, N. Schmitz, C. Rackauckas, R. Heintzmann, Frankschae, A. Noack, C. Lucibello, K. Fischer, A. Robson, Cossio, J. Ling, MattBrzezinski, R. Finnegan, A. Zhabinski, D. Wennberg, M. Besançon, and P. Vertechi (2022). *JuliaDiff/ChainRules.jl: V1.44.7*. Version v1.44.7. Zenodo. DOI: 10.5281/ZENODO.4754896 (cit. on pp. 48, 132).

Wiecek, M. M., M. Ehrgott, and A. Engau (2016). "Continuous Multiobjective Programming". en. In: *Multiple Criteria Decision Analysis: State of the Art Surveys*. Ed. by S. Greco, M. Ehrgott, and J. R. Figueira. International Series in Operations Research & Management Science. New York, NY: Springer, pp. 739–815. ISBN: 978-1-4939-3094-4. DOI: 10.1007/978-1-4939-3094-4_18 (cit. on p. 161).

Wilder, B., B. Dilkina, and M. Tambe (2019). "Melding the Data-Decisions Pipeline: Decision-Focused Learning for Combinatorial Optimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33, pp. 1658–1665. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33011658 (cit. on p. 134).

Yu, J. and S. M. LaValle (2013). "Planning Optimal Paths for Multiple Robots on Graphs". In: *2013 IEEE International Conference on Robotics and Automation*. 2013 IEEE International Conference on Robotics and Automation, pp. 3612–3617. DOI: 10/gf3bxz (cit. on p. 72).

Zarepisheh, M. and E. Khorram (2011). "On the Transformation of Lexicographic Nonlinear Multi-objective Programs to Single Objective Programs". en. In: *Mathematical Methods of Operations Research* 74.2, pp. 217–231. ISSN: 1432-5217. DOI: 10.1007/s00186-011-0360-7 (cit. on p. 162).