Segmentation of facade images with shape priors

Mateusz Kozinski

▶ **To cite this version:**

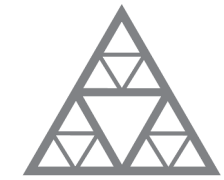
Mateusz Kozinski. Segmentation of facade images with shape priors. Signal and Image processing. Université Paris-Est, 2015. English. NNT : 2015PESC1017 . tel-01240590v2

HAL Id: tel-01240590
https://pastel.hal.science/tel-01240590v2

Submitted on 23 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



UNIVERSITÉ PARIS-EST

Segmentation of Facade Images with Shape Priors

Mateusz KOZIŃSKI

A doctoral thesis, supervised by **Renaud MARLET**.

Submitted to the **École Doctorale Paris-Est Mathématiques et Sciences et Technologies de l'Information et de la Communication**.

To be presented on the 30th of June 2015 to a committee consisting of:

Reviewers	Iasonas	KOKKINOS	École Centrale Paris
	Peter	WONKA	Arizona State University, KAUST
Examiners	Frédéric	JURIE	Université de Caen
	Hayko	RIEMENSCHNEIDER	ETH Zürich
	Hugues	TALBOT	ESIEE Paris
Supervisor	Renaud	MARLET	École des Ponts ParisTech
Invited	Olivier	TOURNAIRE	Centre Scientifique et Technique du Bâtiment

École des Ponts ParisTech
CERTIS/IMAGINE
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Université Paris-Est Marne-la-Vallée
École Doctorale Paris-Est MSTIC
Département Études Doctorales
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77454 Marne-la-Vallé cedex 2
France

Acknowledgements

First, I would like to acknowledge my advisers and mentors from the Imagine laboratory. I am very grateful to my supervisor Renaud Marlet for his warm and open attitude, for believing in me and for his support. He never let me feel intimidated by my limited expertise and was a constant source of encouragement, which helped me to make it through difficult moments of my work. I also thank Olivier Tournaire for his support when co-supervising this thesis. In addition to my supervisors, I would like to express my gratitude to Guillaume Obozinski, whose patience in explaining me the basics of optimization was unlimited. I benefited a lot from his guidance both in terms of technical expertise and general attitude to research.

It is an honor to present this work to the members of my examination committee: Peter Wonka, Iasonas Kokkinos, Frédéric Jurie, Hugues Talbot and Hayko Riemenschneider. I am grateful for their precious time devoted to reviewing this thesis and participating in the defense.

I also thank Andrea Cohen from ETH Zurich and Anđelo Martinović from KU Leuven for sharing the results of their experiments on facade parsing, which made experimental comparison of our methods feasible.

My work would have been much more difficult, and certainly way less entertaining, if not for my colleagues. I owe thanks to David Ok for adopting me to participate in one of his projects. I would not be able to complete the work on adjacency patterns without the cooperation of Raghudeep Gadde and Sergey Zagoruyko. Thank you, Raghudeep and Sergey. I also thank Francisco Vitor Suzano Massa and Amine Bourki for hours of exciting discussions. It made it so much easier to understand many concepts appearing in the recent literature! I received a lot of cordial support from all the members of the Imagine group. In particular, I would like to thank: Pierre Moulon, Victoria Rudakova, Liu Zhe, Alexandre Boulc'h, Ferran Espuny, Achraf Ben Hamadou, Laura Fernàndez Julià, Martin de La Gorce, Marina Vinyes, Praveer Singh, Spyros Gidaris and Bruno Conejo.

To encourage me to pursue my dream, my wife Magda agreed to abandon a well organized life and spend three years on emigration, far from the beloved family and friends. This sacrifice means a lot to me. I thank my parents and my brother Kuba, for their unconditional support.

I am very grateful for the support I received from my best friends: Chi-Wei Chen, Maria Vakalopoulou, Norbert Bus and Ravi Kiran. Meeting you already made taking up the challenge of doctoral studies worth it.

Finally, I would like to acknowledge the funding we got from Centre Scientifique et Technique du Bâtiment. The work presented in this thesis was also partly funded by the ANR project Semapolis ANR-13-CORD-0003.

Abstract

The aim of this work is to propose a framework for facade segmentation with user-defined shape priors. In such a framework, the user specifies a shape prior using a rigorously defined shape prior formalism. The prior expresses a number of hard constraints and a soft preference on spatial configuration of segments, constituting the final segmentation. Existing approaches to the problem are affected by a compromise between the type of constraints, the satisfaction of which can be guaranteed by the segmentation algorithm, and the capability to approximate optimal segmentations consistent with a prior.

In this thesis we explore a number of approaches to facade parsing that combine prior formalism featuring high expressive power, guarantees of conformance of the resulting segmentations to the prior, and effective inference. We evaluate the proposed algorithms on a number of datasets. Since one of our focus points is the accuracy gain resulting from more effective inference algorithms, we perform a fair comparison to existing methods, using the same data term.

Our contributions include a combination of graph grammars for expressing variation of facade structure with graphical models encoding the energy of models of given structures for different positions of facade elements. We also present the first linear formulation of facade parsing with shape priors. Finally, we propose a shape prior formalism that enables formulating the problem of optimal segmentation as the inference in a Markov random field over the standard four-connected grid of pixels. The last method advances the state of the art by combining the flexibility of a user-defined grammar with segmentation accuracy that was reserved for frameworks with pre-defined priors before. It also enables handling occlusions by simultaneously recovering the structure of the occluded facade and segmenting the occluding objects. We believe that it can be extended in many directions, including semantizing three-dimensional point clouds and parsing images of general urban scenes.

Resumé

L'objectif de cette thèse concerne l'analyse automatique d'images de façades de bâtiments à partir de descriptions formelles a priori de formes géométriques. Ces informations définies par un utilisateur permettent de modéliser, de manière formelle, des contraintes spatiales plus ou moins dures quant à la segmentation sémantique produite par le système. Ceci permet de se défaire de deux principaux écueils inhérents aux méthodes d'analyse de façades existantes qui concernent d'une part la garantie que l'algorithme de segmentation respecte bien les contraintes fortes, et d'autre part la capacité à s'approcher d'une segmentation optimale par rapport aux a priori.

Nous proposons d'explorer au travers de cette thèse différentes méthodes alternatives à celles proposées dans la littérature en exploitant un formalisme de représentation d'a priori de haut niveau d'abstraction, les propriétés engendrées par ces nouvelles méthodes ainsi que les outils de résolution mis en œuvre par celles-ci. Le système résultant est évalué tant quantitativement que qualitativement sur de multiples bases de données standards et par le biais d'études comparatives à des approches à l'état de l'art en la matière.

Parmi nos contributions, nous pouvons citer la combinaison du formalisme des grammaires de graphes exprimant les variations architecturales de façades de bâtiments et les modèles graphiques probabilistes modélisant l'énergie attribuée à une configuration paramétrique donnée, dans un schéma d'optimisation par minimisation d'énergie, ainsi qu'une nouvelle approche par programmation linéaire d'analyse avec à priori de formes.

Enfin, nous proposons un formalisme flexible de ces a priori devançant de par ses performances les méthodes à l'état de l'art tout en combinant les avantages de la généricité de contraintes simples écrites par un utilisateur, à ceux de la précision de la segmentation finale qui se faisait jusqu'alors au prix d'un encodage préliminaire restrictif de règles grammaticales complexes propres à une famille architecturale donnée. Le système décrit permet également de traiter avec robustesse des scènes comprenant des objets occultants et pourrait encore être étendu notamment afin de traiter l'extension tri-dimensionnelle de la sémantisation d'environnements urbains sous forme de nuages de points 3D ou d'une analyse multi-image de bâtiments.

Publications of the Author

Ok, D., Koziński, M., Marlet, R., Paragios, N. (2012). High-Level Bottom-Up Cues for Top-Down Parsing of Facade Images. In *Proc. 2nd Joint Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT).*

Koziński, M., Marlet, R. (2014). Image parsing with graph grammars and Markov Random Fields applied to facade analysis. In *Proc. 2014 IEEE Winter Conference on Applications of Computer Vision (WACV).*

Koziński, M., Obozinski, G., Marlet, R. (2014). Beyond Procedural Facade Parsing: Bidirectional Alignment via Linear Programing. In *Proc. 2014 Asian Conference on Computer Vision (ACCV).*

Koziński, M., Gadde, R., Zagoruyko, S., Obozinski, G., Marlet, R. (2015). A MRF Shape Prior for Facade Parsing with Occlusions. In *Proc. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Contents

Chapter 1

Introduction

1.1 Problem definition

The goal of facade parsing is to segment building images into regions corresponding to architectural elements, like windows, balconies and doors. Unlike in the case of general image segmentation, resulting segments have to satisfy structural constraints. Two types of constraints that we find most useful for facade modeling are alignment of facade elements as well as vertical and horizontal order. For example, we could require some windows to be aligned horizontally or vertically, and a balcony, if any, to be below a window. Other types of handy constraints include relative sizes of elements, and number of elements in a facade, like the possible number of floors. In this work we focus on parsing rectified facade images. An example input image and a segmentation produced by a facade parsing algorithm are presented in figure 1.1.

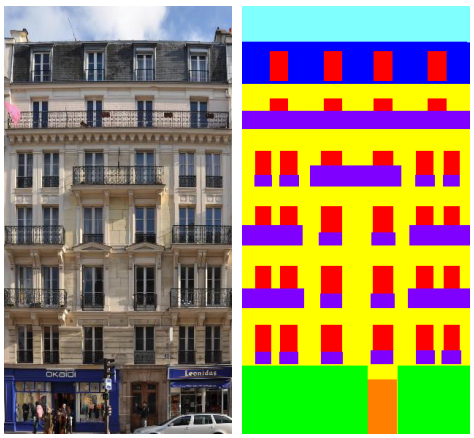


Figure 1.1: Left: an image from the ECP dataset [59]. Right: result of parsing the image on the left. Colors correspond to different semantic classes.

In this thesis we address the problem of facade parsing with prior knowledge. In general, a prior can be specified by a user and/or learned automatically from training examples, and expresses hard constraints on facade structure and soft preference for some patterns of facade elements over others. The resulting segmentation has to conform to the prior, that is, satisfy the hard constraints specified therein and

minimize an objective function consisting of a data term and a term expressing the soft preference. We assume arrangements of facade elements vary considerably, but the patterns of variation are known, so that it is possible to produce a concise specification of valid facade structures for a given architectural style. Such specification can constitute a prior encoding possible patterns of facade elements. For example, the facade in figure 1.1 is an instance of the Haussmannian style. Buildings of this style have five or six floors, of which the top and the second floor are very likely to have balconies running through the whole facade, and balconies in the remaining floors can be misaligned vertically. Windows in a Haussmannian facade are arranged in a grid pattern, with the exception of the top floor, which can contain windows misaligned with ones in the rest of the facade but aligned with attic windows. A ground floor usually contains retail and service facilities, while the upper floors have a residential character.

1.2 Application context

While this work is focused on parsing individual rectified images of building facades, it is in line with the long term goal of creating models of existing buildings from images. Such models could serve the purpose of urban planning, be useful for documenting existing old buildings and planning renovations. Realistic models of urban scenes also find applications in computer games. Additionally, there are many applications within the civil engineering industry, including simulations of shadow casting, thermal performance, noise and air pollution propagation in the city, or prediction of electric energy yield from solar panels placed on building roofs. Models required for many of these applications should contain semantic information. For example, for architectural planning it is useful to have a wire frame model resembling a drawing produced in a Computer Aided Design (CAD) program by an architect, as opposed to a model consisting of large number of irregular polygons, containing noise and missing information, typically resulting from 3D reconstruction from images. For thermal simulations it is crucial to know the openings in the building structure, like doors and windows. Semantic information would also help in realistic rendering of models created from images by providing information about the material in different parts of the model, for example brick (wall), glass (window) and metal (roof). A low number of polygons is a desired feature of models used for all the listed applications.

The problem of obtaining 3D scene models from multiple images is broadly studied in the computer vision community. However, a typical result of a 3D reconstruction pipeline is a mesh consisting of a large number of polygons, with noise, holes resulting from missing information and without semantic information. Such 'unstructured' models can be used for creating visualizations of urban scenes, but are of limited utility for planning renovations or conducting thermal simulations. The difference is presented in figure 1.2, which contains an output from a 3D reconstruction pipeline and a 'structured' model of perfect geometry. It is evident that the models are very different, even though the semantic information contained in the 'structured' model is not visualized.

The current state of art in facade parsing does not allow to directly address the

Figure 1.2: An output of a 3D reconstruction pipeline (left) and a geometric building model (right). The model contains a low number of polygons and is free from noise. Additionally, it features perfect alignment of architectural elements. We are interested in models that contain semantic information (not presented in the image) in addition to geometry.

problem of reconstruction of 3D models from multiple images, or parsing general images of urban scenes. Existing facade parsing frameworks impose certain constraints on input data. The algorithms are restricted to parsing single rectified images of building facades. However, facade parsing could be useful in real life applications. There has been research on creating 3D models of facades from single images [41, 39]. Other work [59, 34] demonstrated creating simple 3D building models based on facade segmentations, by assigning depth to segments based on their semantic labels. There exist algorithms [33, 9] which, unlike the first attempts at facade parsing [28, 60], can handle non-rectangular shapes of facade elements and occlusions. In this work we advance the state of the art of facade parsing with user-defined shape priors towards applicability by enabling explicit modeling of occlusions and irregularly shaped facade elements.

A possible application scenario involves acquiring facade images by a side-looking vehicle mounted cameras and rectifying them manually or automatically [3]. The resulting rectified images of building facades can be parsed and 3D models can be created from the segmentations by assigning class-dependent depth to each segment. Some applications, involving collection of information about urban environment, can utilize the segmentations directly. For example, estimating the area of glass in a building facade is necessary to assess thermal performance of a building, and the estimated number (and type) of shops and service facilities can be used to statistically characterize urban scenes. In another application scenario a user manually defines the rough shoe-box geometry of a building visible in an image, in such a way that faces correspond to distinct building facades. These regions can then be parsed resulting in a segmentation that can be used to increase the level of detail of the model.

Finally, we think the algorithms presented in this thesis can be generalized to more challenging tasks, like parsing images of urban scenes without the rectification constraints, or semantizing three-dimensional point clouds representing urban scenes. The advantage of models resulting from parsing over the ones created by traditional 3D reconstruction, would be the presence of the semantic information

and satisfaction of structural constraints defined in the prior. Some attempts at parsing unrectified images and point clouds of urban scenes have already been made [4, 52].

1.3 Challenges associated to the problem

Conceptually, the stated problem can be seen as consisting of three research questions: 1) how to specify shape priors, 2) how to parametrize a segmentation consistent with a prior, 3) how to obtain an optimal segmentation consistent with the specified prior. Obviously, none of these questions can be answered independently of the others, and this entanglement is one of the main difficulties of the problem. For example, there exists a trade-off between the expressive power of a prior formalism and the capability of approximating optimal segmentations consistent with a prior. In section 2 we list a number of previous works in which the advantage of a powerful and concise prior formalism is counterbalanced by the necessity of random exploration of a large space of segmentations for inference. In other algorithms the parametrization of the segmentation enables efficient search for optimal segmentations, but limits the types of handled constraints, or fails to guarantee the global satisfaction of the constraints.

Another challenge is the highly connected nature of facade models. Any attempt to parametrize the facade presented in figure 1.1, whether in terms of positions of windows, or their corners, or snaplines, results in a highly connected model with a loopy structure. That is, if we express the model in form of a graph, where nodes correspond to variables and each (hyper-) edge connects variables involved in some constraint, the graph is going to have loops. Such models make the search for optimal segmentations difficult. The simplest example of such loopy constraints is the horizontal and vertical alignment of windows.

Finally, facades feature structural variation. In other words, we do not know a priori how many elements are present in a facade, or how they are entangled by the constraints. For example, we do not know how many floors or window columns there are in a given facade. In consequence, we cannot expect that a model with a fixed number of variables encoding positions of architectural elements, like windows and balconies, will be capable of representing facades of different structures.

1.4 Contribution

In this work we address the problem of facade parsing in its three aspects. First, we propose a shape prior specification that enables encoding simultaneous alignment of facade elements, their order and non-overlap. Second, we provide a parametrization of facade segmentation that represents facades as points in a space where structural constraints are satisfied, or where it is easy to enforce their satisfaction, and which facilitates evaluating the prior energy term encoding soft preference on facade layout. Third, we present an efficient optimization algorithm that can guarantee satisfaction of the constraints constituting a prior. Our main goal is to guarantee 'structural correctness' of the resulting segmentations, that is, their conformance to prior, while

proposing efficient optimization scheme. We present three different solutions, developed in pursuit of this goal, where each consecutive formulation is more easily applicable, versatile and faster than the previous one.

The main technical contributions are:

- an image parsing framework combining graph grammars with MAP-MRF optimization, where the graph grammar models structural variation, and the MAP-MRF optimization provides an efficient way of inferring positions of objects for a given tentative structure of the scene;
- a binary linear formulation of the parsing problem, which is the first formulation with user-defined grammar, where the optimal segmentation can be inferred without resorting to sampling segmentations or subsampling the image;
- an adjacency pattern-based approach to facade parsing, which is a state of the art in facade parsing, and the first prior-based algorithm where occlusions and irregular shapes of facade elements can be modeled.

The adjacency pattern-based framework is the first prior-driven algorithm that enables modeling occlusions and can simultaneously recover both the structure of the underlying facade and the boundary of the occluding object. This is an important step towards real life applications of facade parsing, as occlusions are often encountered in practice. The framework was extensively tested on four different datasets and was shown to consistently yield better segmentations than competing methods while using the same image cues.

Chapter 2

Existing Work

2.1 Grammar-type shape priors for buildings

Shape grammars are particularly popular as shape priors for modeling buildings. They can be used both to model geometry of structures in three dimensions and to encode two-dimensional arrangements of facade elements. They are generative models resembling string grammars used in formal and natural language processing. A grammar $\mathcal{G} = (\mathcal{S}, \mathcal{T}, \mathcal{N}, \mathcal{P})$ consists of a set of terminal symbols $\mathcal{T}$, a set of nonterminal symbols $\mathcal{N}$, a set of productions (also called production rules) $\mathcal{P}$, and a starting symbol, or axiom, $\mathcal{S} \in \mathcal{N}$. A production $p \in \mathcal{P}$, denoted $p : N \longrightarrow (n_1, \ldots, n_k)$, is a mapping of a nonterminal symbol $N \in \mathcal{N}$, called the left-hand side of the production, to a (possibly structured) collection of terminals and nonterminals $(n_1, \ldots, n_k)$, where $n_1, \ldots, n_k \in \mathcal{N} \cup \mathcal{T}$. The collection of symbols $(n_1, \ldots, n_k)$ is referred to as the right-hand side of the production. In shape grammars, a symbol corresponds to a basic shape, and a collection of symbols represents a complex shape, which is a spatially structured set of basic shapes. The generation of a collection of terminal symbols from a grammar is called derivation. A basic step of the derivation process consists in applying a production, which substitutes a nonterminal symbol in the collection, that matches the left-hand side of the production, with the right-hand side of the same production. In the case of shape grammars, each production application substitutes one of the nonterminal basic shapes in the derived complex shape by a spatially structured combination of terminal and nonterminal basic shapes, according to the applied production. The derivation starts with a collection consisting only of the starting symbol $\mathcal{S}$ and develops the collection by recursively applying productions to nonterminal symbols that constitute it. The process continues until the collection contains only terminal symbols. The choice of productions that can be applied to each nonterminal can result in a very large, possibly infinite, set of collections that can be derived from the grammar. A derivation from a grammar can be represented by a parse tree. The nodes of a parse tree represent grammar symbols. The root represents the starting symbol and a link from parent to child exists between a nonterminal and each of the symbols constituting the right-hand side of the production applied to substitute the nonterminal.

Split grammars are a particular form of shape grammars, where the basic shapes are rectangular image regions, and where a production splits a nonterminal rectan-

gular region along one of coordinate axes. We provide an example of a split grammar and an illustration of the derivation process in section 2.1.2.

2.1.1 Grammars for generating models of urban scenes

The generative approach proved to be convenient for creating models of buildings and urban scenes. In their work [38], Müller et al. use the concept of shape grammar to represent a collection of building shapes. Each symbol is assigned a number of attributes, including a three-dimensional cuboid containing the corresponding basic shape, called a scope of the shape. A production can be conditional, that is, it can contain conditions on the attributes of the substituted nonterminal, under which the production is applicable. A constraint on the attributes of the substituted nonterminal and the configuration of basic shapes that is inserted in its place can also be specified in a production. This is used to avoid generating structurally invalid buildings, for example, ones where elements generated from different nonterminals overlap. By randomly selecting the productions and attribute values during derivation, the grammar can be used to generate building models of various structural configurations. The grammar contains operations on three-dimensional shapes, and on their two-dimensional faces. Arguably the most important type of productions is a split rule. It subdivides a scope of an atomic shape along one of its coordinate axes, resulting in a number of new atomic shapes. This type of rule is of particular interest for modeling buildings as it enables encoding crucial constraints: alignment, non-overlap and order of elements.

2.1.2 Facade parsing with grammar-based shape priors

The modeling of buildings on the basis of images has been a popular field [4, 72, 41, 20] due to its numerous applications. Inspired by the work of Müller et al., some research was aimed at applying the grammar formalism to retrieving building models from images [28, 60, 59, 36, 47]. The user specifies a shape grammar and proposed algorithms try to find a sequence of instantiated grammar rules yielding an optimal segmentation.

The quality of the segmentation is measured by means of a merit function defined for each pixel of the image. The function models the likelihood that a point belongs to a region representing an architectural element of a given category. The merit function is typically learned from training data. In their work, Teboul et al. [60, 59] used a random forest [7] to estimate the class likelihoods. A more accurate merit has been used by Martinovic et al. [33]. It combines an image segmentation obtained by means of a Recursive Neural Network [53] with results of object detection. Cohen et al. [9] propose yet another merit function: a multi-feature extension of TextonBoost [29]. The used features include SIFT, ColorSIFT, Local Binary Patterns and location features. Feature vectors are clustered to create dictionary entries and the final feature vector is a concatenation of histograms of appearance of cluster members in a neighborhood of 200 randomly sampled rectangles. The per-pixel energies are output by a multi-class boosting classifier [50]. An unstructured image segmentation can be obtained by assigning the most likely class to each pixel independently. A

shape prior can be seen as a regularizer. In this work we show that the constraints encoded in the prior not only ensure the 'semantic correctness' of the segmentation, but also improve the classification results. In addition to 'hard' constraints, the prior can also contain a 'soft' component, that expresses the a priori likelihood of segmentations. The goal of facade parsing is to find a parse (a derivation tree and the resulting segmentation of the input image) that maximizes the a posteriori likelihood of the data given the model.

Teboul et al. [60], propose an algorithm that segments rectified images of building facades based on a binary split grammar specified by the user. A split grammar is a grammar where all the productions are splits, and the adjective 'binary' indicates that each production can yield at most two child symbols. The symbols represent rectangular image regions and a production splits a region along one of the image axes into two child rectangles. A split grammar is a concise manner of encoding the order and alignment constraints in one direction. Split grammars can be seen as attribute grammars, where each symbol is assigned a number of attributes, including the dimensions of the corresponding rectangle, and where the productions define possible configurations of attributes of the child symbols given the attributes of the parent. In particular, a production can encode the possible sizes of the child regions. We present a simple two-dimensional split grammar in table 2.1 together with an example derivation in figure 2.1. For simplicity we neglect the attributes of productions, including sizes of shapes, in the example.

Parsing, that is, segmenting an image according to a grammar, turns out to be much more difficult than generating a plausible arbitrary segmentation from a grammar. There exist two approaches to parsing string grammars that could hypothetically be adapted to parsing shape grammars. Unfortunately, both paradigms have strong disadvantages when applied to image parsing. A top-down parsing algorithm attempts to construct a parse tree by starting with the root, and applying productions until first terminal symbols are generated, which are then matched to the input. A bottom-up parsing algorithm tries to construct a parse tree by first initiating the leaves based on the input and then grouping the leaves into collections that correspond to right-hand sides of productions and assigning each of them a parent. The process continues recursively until the tree is terminated with a starting symbol and contains all the input symbols. On the abstract level, there are two general problems with using such algorithms for parsing images. First, images are not composed of unambiguously identified symbols. There is an uncertainty associated to the class of each pixel. In computer vision this problem is typically handled by combining a merit function with a regularization term expressing the prior likelihood of the result. Then a segmentation is sought which optimizes a combination of both terms. Parsing algorithms work on unambiguously defined symbols and do not handle sufficient level of uncertainty. Second, parsing algorithms do not handle missing or false positive symbols to an extent sufficient for treating images. This eliminates the possibility of running a traditional parsing algorithm on primitives detected in an image. On the other hand, when pixels are treated as individual symbols, the computational intensity of parsing becomes prohibitive. These problems are well exemplified by existing facade parsing frameworks, presented below.

Teboul et al. [60, 59], tried to adapt the top-down parsing principle to search for

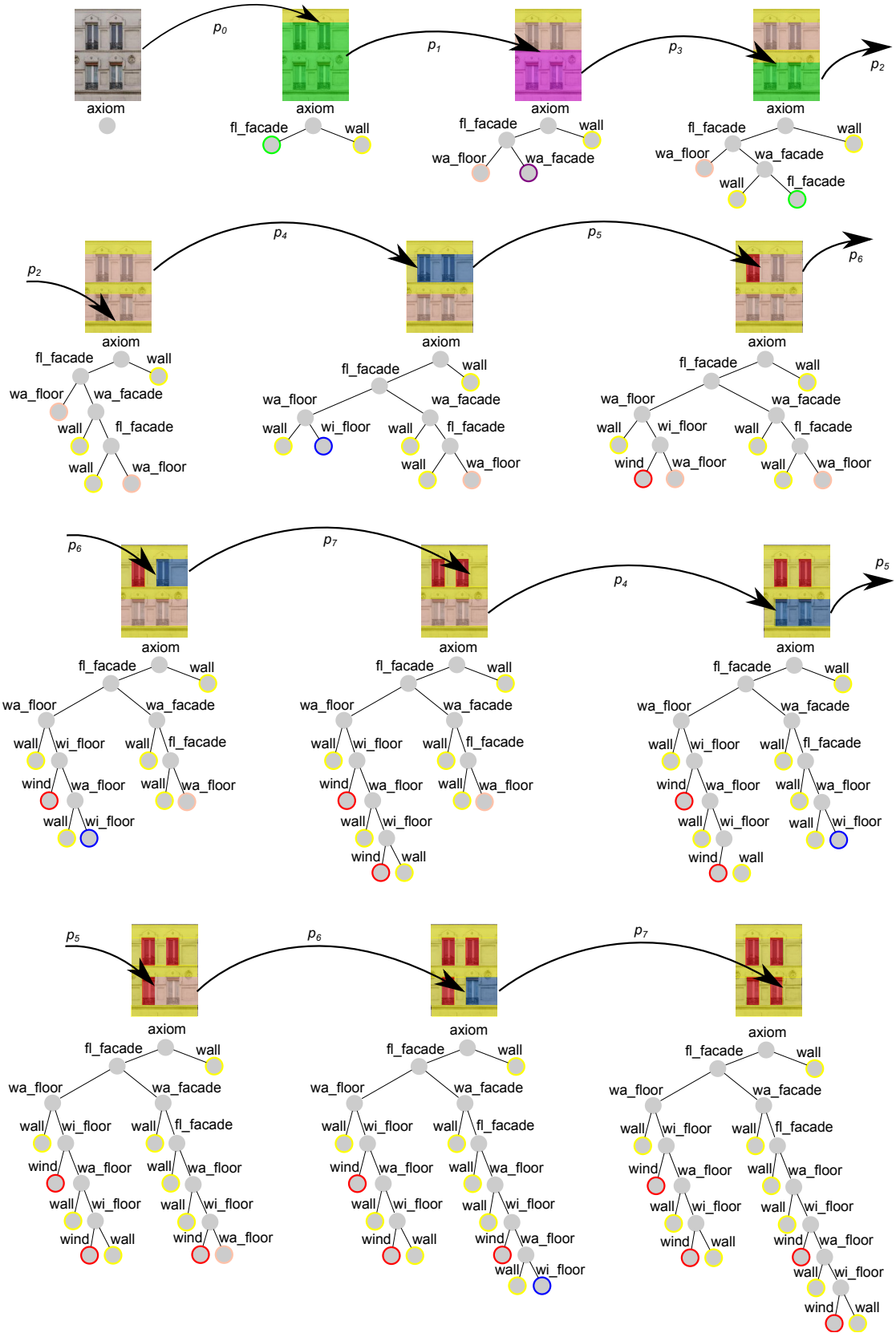
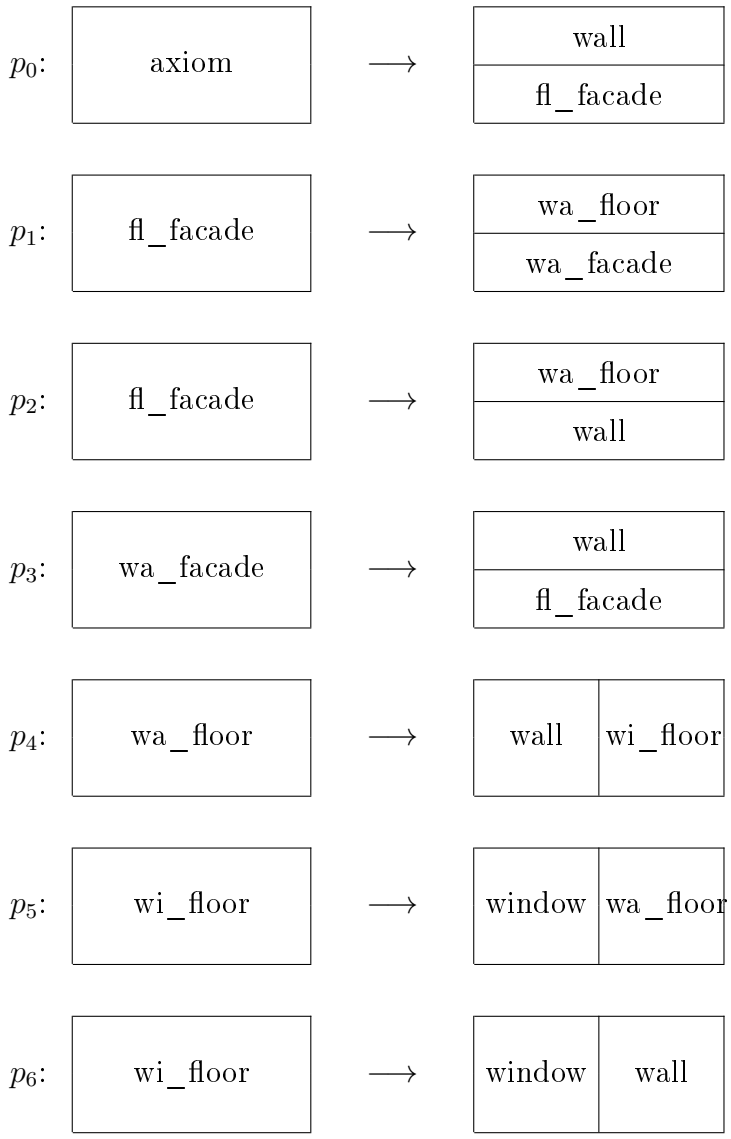


Figure 2.1: A derivation from the grammar presented in table 2.1, overlaid on the corresponding facade image. The black arrows between images denote applications of productions. Parse trees corresponding to each step of the derivation are placed under the images.

Table 2.1: A simple set of grammar rules, modeling a binary facade consisting of windows and wall tiles. The adopted naming convention is that, if the name of a nonterminal symbol is complex, the first part denotes the topmost, or leftmost symbol that can be derived from it. For example, 'fl_facade' denotes a facade region that has a floor at the top.



optimal segmentations. However, the cost of applying a production to a nonterminal is not known until all the terminals descending from it have been derived. Moreover, the search space consists of all the parse trees that can be generated from a given grammar and is therefore very large. The authors resort to a random walk procedure for exploring the space of shapes, represented as derivation trees. In another work [59], the same authors present an improved optimization algorithm, based on the Q-learning paradigm, that explores the space more efficiently. However, the exploration remains random in principle resulting in potentially large variations of the final segmentation. Therefore it cannot be relied on to produce optimal, repeatable results.

In appendix A we present an attempt to address the limitation of the top-down

parser [60, 59] by guiding the exploration of the space of parse trees with strong bottom-up cues. We use window detections in addition to texture classification and design a prior on the split positions based on detected line segments, as opposed to the gradients used in the original algorithm. Even though the resulting algorithm outperforms the original parser, it suffers from the same problems caused by inference based on random exploration of the problem space.

Some research has been targeted at using the bottom-up algorithms, similar to ones used in string grammar parsing, for facade parsing [47]. Unfortunately, they feature a prohibitive computational complexity on images, partly due to the two dimensional nature of images, and in some part because individual pixels or image segments cannot be classified unambiguously, unlike symbols in a string grammar. Riemenschneider et al. [47] use a grammar defined on a two-dimensional grid and a 2D version of the Coke-Younger-Kasami (CYK) parser to find the optimal segmentation. However, its complexity is $O(w^2h^2N)$, where w and h are the dimensions of the image and N is the number of possible combinations of production rule attributes (including splitting positions). This limits practical applications of the algorithm to grids of about 60 by 60 cells. To circumvent this limitation the authors explore different methods of selecting the possible split positions.

Another issue with split grammars is the difficulty of encoding simultaneous alignment in two dimensions. For example, two floors in figure 2.1 are derived independently and the basic grammar formalism does not provide a mechanism for enforcing the alignment. In their work [60, 59] Teboul et al. allow the user to specify, when writing a grammar, which symbols should be split in exactly the same way. This introduces hidden dependencies in the parse tree and the derivation process.

Summarizing, the split grammar-based methods provide a neat way of encoding shape priors. The segmentation is parametrized as a parse tree, encoding the applied productions and attributes of resulting shapes. However, the existing frameworks fail to offer a reliable and efficient optimization scheme. Due to the very large search space the algorithms suffer from the 'curse of structural exploration'. They search the space of parse trees more or less randomly [60, 59], which does not guarantee optimality nor repeatability, sometimes severely subsampling the image [47].

2.1.3 Grammar learning

Learning shape grammars from labeled images is an interesting problem that we do not handle in this work. However, we mention it here due to the connection between parsing and grammar learning.

Martinović and Van Gool propose to learn 2D split grammars by Bayesian model merging [34]. The algorithm starts with one grammar per each image in the training set. These are then merged to form a single grammar, by giving them all the same starting symbol, and aggregating the symbols and productions. Then, pairs of nonterminals are merged in an iterative process, greedily optimizing an energy containing a data term, preferring grammars with more precise segmentations, and a regularization term, promoting more concise grammars. The authors also propose a reversible jump Markov chain Monte Carlo algorithm for inference with the learned grammar.

Gadde, Marlet and Paragios [16] proposed another grammar learning algorithm. Its aim is to learn a grammar specific to an architectural style, given a generic grammar and a number of labeled images. First, the ground truth images are parsed using the generic grammar, yielding ground truth parse trees. Then, recursive applications of the same rule are compressed, to facilitate the following step, identifying common subtrees. The next steps clusters large subtrees that are common to many parse trees. Linear programming-based clustering is used. The clusters are represented as individual rules. This means the new rules can be complex, consisting of combinations of vertical and horizontal splits. This considerably speeds up convergence of the parsing algorithm.

A co-segmentation algorithm proposed by Martinović and Van Gool [35] is another attempt to eliminate the requirement of specifying a shape grammar. It operates on labelings of facade images obtained by a general-purpose segmentation algorithm. It processes all the test images at the same time. The algorithm is based on recursive decomposition of rectangular image regions by splitting them along vertical or horizontal direction. The decomposition step has a form of co-segmentation. It processes a cluster of image regions (possibly originating from different images) at the same time, by maximizing a combination of the scores specific to individual regions and the score expressing the consistency of resulting segments between the regions. Then, the regions resulting from the decomposition are clustered and the decomposition is applied to the new clusters. Although the framework produces inferior results to ones obtained by parsing algorithms, it addresses the interesting problem of inferring the semantically correct labelings and the underlying constraints in the same time. The hierarchical facade representations created in the process have been shown to aid facade retrieval and enable generation of facade models.

The problem of grammar inference has been addressed by Weissenberg et al. [67] from the point of view of facade compression, comparison and synthesis. The algorithm accepts annotations of rectified facade images on input, which are then transformed into parse trees by repetitively applying splits that extremize a handcrafted energy. Grammar productions are formed by processing the parse trees bottom-up and aggregating splits. In particular, each sequence of consecutive splits in the same direction is aggregated to form a single production. Groups of splits in different tree branches, and different trees, that are similar in terms of their direction and classes of children also give rise to new production rules. The authors propose a distance function defined on the parse trees, that can be used for facade comparison and retrieval.

A related problem is inferring a minimal grammar that encodes a given facade layout. Its applications include compression of building models and learning grammars from examples. The algorithm by Wu et al. [69] attempts to find a deterministic split grammar that produces a given facade layout and that minimizes a predefined cost function. The method combines greedy and random exploration of the space of grammars that produce a given facade layout, generating a large number of grammars. It keeps the grammar that scores the best with respect to a predefined cost function, which evaluates the complexity of each used production. The resulting grammar can be modified by changing the number of repetitions of architectural elements, and their sizes, or by combining a number of grammars obtained from

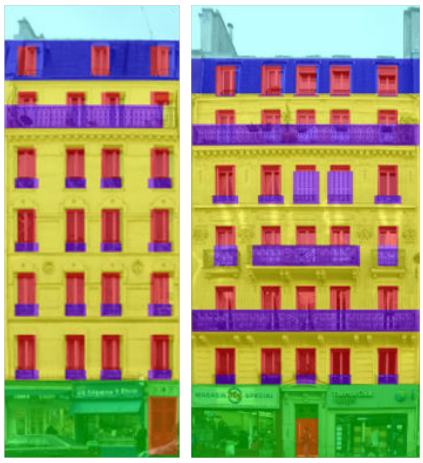


Figure 2.2: Semantically incorrect facade segmentation that can result from applying locally optimal corrections [33], for example, aligning pairs of lines that are sufficiently close. Note the balcony ending in the middle of a window in the left image, and the excessively large balconies in the right image. These artifacts result from not enforcing hard constraints on the resulting segmentations.

more than one facade. This way new facade layouts can be generated.

2.2 Other methods of facade parsing

Some facade segmentation methods do not use user-defined shape priors. One alternative approach to the problem is to enforce the structural constraints on results of a general-purpose segmentation algorithm. In [33], Martinovic et al. combine results of a Recursive Neural Network with object detections to form unary potentials of a Markov Random Field encoding an initial image segmentation. Then 'soft' architectural principles are applied as a postprocessing step after image segmentation. The initial segmentation is modified to satisfy a number of 'weak architectural principles': some elements are given rectangular shapes; rectangles, boundaries of which are sufficiently close, are aligned; doors are inserted into the lower parts of facades. The method cannot accommodate 'hard' structural constraints. Besides, the set of 'architectural principles' is different for each dataset and no formal way of specifying them has been proposed. Moreover, applying local corrections to a segmentation, for example, aligning lines that are close enough, does not necessarily yield a semantically correct segmentation. The method can produce artifacts, like a balcony ending in the middle of its corresponding window (see figure 2.2).

A more recent work by Cohen et al. [9] uses a sequence of dynamic programs (DPs) to recover a segmentation that respects a set of hard-coded constraints and attains state-of-the-art performance on the standard datasets. Each DP makes the current labeling more detailed. The first one operates along the vertical axis and identifies the floors. The following ones identify windows in each of the floors, the boundary between the sky and roof, and the door and shop. This configuration of DPs encodes a particular structure of the facade, which should be horizontally decomposable into floors, which themselves can be decomposed into sequences of windows and wall tiles. If the actual facade configuration is different, the algorithm approximates it with a segmentation that conforms to the structure encoded by the

DPs. Also in this case there is no systematic way for the user to encode different sequences of DPs that would correspond to different architectural styles. As the DPs operate independently over image regions, the approach does not enforce or promote simultaneous alignment of shapes in two dimensions.

The work of Yang et al. [70] focuses on the application of rank-1 matrix approximation to facade parsing. A binary classifier of window color is applied to the facade image, and the resulting image is approximated as a rank-1 matrix. To handle facade patterns more complex than a single grid, the algorithm randomly divides the image into a number of rectangular regions and then operates on these regions separately, representing them as rank-1 matrices. The procedure is repeated a number of times and the segmentation that scores best against the per-pixel classification is kept as the final result. The main drawback of the algorithm is the limitation to a two-class (window and wall) facades and the lack of a principled method for handling shapes more complex than a grid.

Mathias et al. [36] propose to use the grammar of [38] and generate the building while estimating the attributes of the applied grammar rules from the input images and a 3D point cloud. While the general idea seems attractive, the algorithm has not been shown to perform well with more than two classes of terminal symbols and accommodates only a small subset of rules of the original grammar.

Dai et al. [10] proposed an algorithm in which the input image is first segmented into rectangular regions, then a random field is created over the resulting segments and finally the optimal labeling is obtained by sampling using Swendsen-Wang cuts. The algorithm does not require any user-defined shape specification. Instead, the weights corresponding to potentials promoting alignment of shapes, rectangular shape of some facade objects, similarity of objects of the same class and frequency of occurrence of different classes are learned. The presence of higher-order potentials constrains the spectrum of applicable inference algorithms and the authors resort to sampling. Moreover, learning parameters of the potentials does not guarantee 'semantic correctness' of the results.

Müller et al. [39] propose a method for obtaining semantic, three-dimensional models of building facades from rectified facade images. First, a mutual information criterion is used to subdivide the image into floors and tiles. An 'irreducible facade' is created by grouping the floors and columns that share similar appearance. Then, the tiles are subdivided further. Finally, the semantic class of the rectangular subregions is detected and a facade model is created by tiling three dimensional models of elements of the corresponding classes, stored in a database built a priori. The main limitation of the algorithm is that it requires a facade to follow the fixed decomposition into rows and columns.

2.3 Other grammar-based image models

In this section we discuss grammar-based image models in applications different than facade parsing. Although they are known under the common name of image, or shape grammars, the concepts often differ significantly from grammars used for facade parsing.

Using the prior knowledge of likely compositions of objects present in the scenes

is believed to be beneficial not only for facade parsing, but also for many other vision problems. Although the concept of shape grammars dates back to the 70's and the work of Stiny et al. [55], the idea of representing the image contents in a hierarchical and semantized manner can be traced back to the work of Kanade and Ohta [42, 43]. However, the practical applications of grammars to image interpretation or segmentation are attributed to more recent works [18, 66, 21, 1].

A considerable research effort has been devoted to explore the hierarchical and regular structure of man-made objects in order to improve segmentation or detection results [66, 21, 1]. We focus on flexible grammars that allow the user to encode specific knowledge of the domain in the form of production rules that constrain the space of feasible solutions. The grammar-based image interpretation paradigm is thoroughly reviewed in the work of Zhu and Mumford [73]. A good example of this approach is the rectangle-based grammar of Han and Zhu [18], in which the prior knowledge is represented by means of an and/or graph. The terminal symbols are rectangles and the production rules combine them into rows, columns or grids, and allow for rectangle nesting. This case illustrates one of the main difficulties of the problem: the number of terminals in the solution is unknown. The greedy algorithm presented in the paper copes well with this difficulty. However, since there is no semantic interpretation associated with the rectangles, there is no sensible way of deciding which of any two candidate parse trees is better.

Grammars and part-based models have been combined in application to object detection in [17, 51]. The grammars are hierarchical. That is, the position of a part is dependent on the position of its 'parent' object and the applied production rule. Unfortunately, the structure of certain objects cannot be organized in a hierarchical manner. In the case of facades, windows are aligned horizontally within floors and vertically between different floors. A tree-shaped hierarchy would fail to preserve either the vertical or the horizontal alignment.

2.4 Graphical models

A number of problems formulated in this thesis have the form of finding the optimal configuration of a graphical model [65]. Although we do not address optimization in the framework of graphical models, we use some well known optimization algorithms from the field. In this section we want to highlight the existence of a wide range of algorithms that can be used in this setting. The reader is referred to the survey [22] by Kappes et al. for a detailed comparison.

Graphical models are a powerful tool for expressing probability distributions defined on many variables. They are often represented in form of graphs, where nodes $v \in \mathcal{V}$ represent variables x_v, and (hyper-) edges $e \in \mathcal{E}$ represent potentials defined in terms of the variables represented by nodes that they connect. An energy function is defined over a graphical model, consisting of potentials defined in terms of the variables assigned to nodes and vectors consisting of variables connected by edges

$$E_{\mathrm{MRF}} = \sum_{v \in \mathcal{V}} \phi_v(x_v) + \sum_{e \in \mathcal{E}} \phi_e(\mathbf{x}_e) \ , \tag{2.1}$$

where $\mathbf{x}_e$ denotes a vector of variables connected by edge e. The main idea is that the

structure of the graph encodes conditional independence of variables: any variable is conditionally independent of all its non-neighbors, given the state of its neighbors. In result, functions of many variables, that can be factorized into a sum of factors with small support, can be concisely represented in terms of the factors of smaller dimension. This is especially useful in modeling energy functions in the domain of images, which can have millions of pixels. One common structure of a graphical model is a 4-connected grid of variables corresponding to pixels, in which each pixel is connected to the pixels immediately above, below, to the right and to the left by pairwise potentials. Another model common in computer vision, the Deformable Parts Model (DPM), is used for tracking or detecting objects of known structure in images. In this model a node of the graph corresponds to a variable defining position of a part of an object in the image. The pairwise potentials connecting the parts express the likelihood of their relative position.

In computer vision we are usually looking for a configuration of the variables that minimizes E_{MRF}. This task is called inference. In settings, where some of the potentials correspond to prior likelihoods and others encode conditional likelihoods of the observations given the state of the variables, the labeling that minimizes the energy is called the maximum a posteriori configuration (MAP) and the process is often referred to as MAP-inference.

MAP inference in graphical models with structure that does not contain loops (that is, in trees or chains) can be performed exactly by message passing [40]. Chains can also be solved by dynamic programming using the Viterbi algorithm [63, 14].

However, the models handled in this thesis are highly connected (loopy) and involve discrete variables that can take many labels. These can be addressed using loopy belief propagation [40]. The algorithm is not guaranteed to converge. Provably better minima are attained using Sequential Tree-Reweighted message passing (TRWS) [24]. The third alternative is the dual decomposition algorithm [26, 54].

There exists a class of problems that can be efficiently solved using the so called move-making methods [6, 56]. In particular, binary problems with submodular pairwise potential can be solved exactly. However, our problems do not possess these favorable properties.

Chapter 3

Limiting Sampling to Inference of Facade Structure by Combining Graph Grammars and Graphical Models

3.1 Introduction

As shown in chapter 2.1.2, in existing parsing frameworks, in which the shape prior is encoded in a form of split grammar, a segmentation is represented as a parse tree. Such a parse tree combines two types of parameters: the choice of rule applied to each nonterminal and the split positions. Formulating the problem of finding the optimal parse tree is affected by the following difficulties: 1) for practical grammars, the space of all possible parse trees has a very high dimensionality; 2) the number of variables representing split positions and rule types can vary between different parse trees; 3) the problem requires a discrete formulation, where each variable affects many terms of the objective function and many constraints. The last problem stems from the fact that the energy is a sum of per-pixel potentials. A class for each pixel is induced by the terminal symbol (rectangular region) that overlaps it. The energy contributed by a single rectangular region depends on the position of its four sides, created by splits encoded by four different nodes in the parse tree. A single split often encodes the edge of many rectangular regions, see for example the top, or bottom floor boundary in figure 2.1. Such discrete functions with a large number of highly entangled variables are difficult to optimize. In consequence, all existing grammar-based image parsing methods [28, 60, 59, 52, 47, 44, 18] described in section 2.1, suffer from the lack of a reliable optimization algorithm for yielding optimal derivations from a grammar. For inference, the authors resort to random exploration of the problem space [28, 60, 59, 52, 44], which gives suboptimal results and features high variability of results when run several times. In appendix A we show that improving the bottom-up cues used in the random exploration algorithm improves parsing accuracy only slightly. Other algorithms severely subsample the image to make it suitable for combinatorial processing [47], or apply greedy procedures for building the parse tree [18].

In spite of the lack of reliable optimization algorithms, the grammar-based methods have an advantage over the algorithms that do not use user-defined grammars [33, 10, 9] in that they can guarantee conformance of the resulting segmentations to the constraints specified by the user. We therefore address the general drawback of grammar-based methods in an effort to combine the guarantees of satisfaction of constraints defined in a shape prior with the high performance of methods that do not take user-specified grammars on input. As opposed to split grammar parsing algorithms, that simultaneously sample the type of applied productions and the split positions, we propose to model and optimize the structure of the facade and the position of its parts separately. We represent a structure of a particular facade as a graph. We encode all possible facade structures, representing a single architectural style, in a graph grammar. We sample structures from this grammar. For each structure, the positions of the parts are inferred using standard techniques for determining most likely configurations of graphical models. This way, we limit the application of the less reliable randomized optimization algorithms to structure inference only.

3.1.1 Related work

The problem related to the lack of reliable inference in facade parsing frameworks with user-defined grammars has been exposed in sections 2.1.2 and above. We limit this section to work that is related to the algorithm we propose, which combines graph grammars and graphical models.

Our work is related to the part-based approach to object modeling, that has proven effective in numerous applications including face detection and pose tracking [13, 11]. In this framework, an object is represented as a collection of parts. Detecting, or tracking the object consists in identifying the pose of all its parts in images. The method is based on optimizing an energy function defined on the vector of part poses. The energy consists of unary potentials evaluating the pose hypotheses for each part of the model and pairwise potentials evaluating the likelihood of pairs of poses of certain parts. Optimal part poses can be calculated efficiently if the model has a tree structure. Although in this work we represent the imaged objects as factor graphs, the graphs are not fixed but generated from a graph grammar. They also feature a highly connected and loopy structure.

Grammars and part-based models have been combined in applications to object detection [17, 51]. Girshick, Felzenszwalb and McAllester [17] propose a part-based model for pedestrian detector in which occlusions are handled by variations of the model, encoded by a grammar. The models derived from the grammar are hierarchical: the position of a part is dependent on the position of its 'parent' object and the applied production rule. Unfortunately, the structure of certain objects cannot be organized in a purely hierarchical manner. In the case of facades, windows are aligned horizontally within floors and vertically between different floors. Additionally, windows in the same floor cannot overlap. A tree-shaped hierarchy, where the position of each window depends only on the bounding box of the floor, without interactions between individual windows, would fail to model these spatial interactions.

We were also inspired by the context-sensitive graph grammar parser by Rekers and Schurr [46]. Given a grammar defined on graphs, where production rules substitute graph nodes with subgraphs, and given an input graph, the parser can determine the sequence of graph grammar productions used to derive this input graph, or decide that the graph has not been generated from the grammar. The limitation of the parser is that it cannot handle faults in graph structure, like missing or extra edges or nodes, and that it does not handle uncertainty, for example nodes and edges hypothesized with some probability. In computer vision applications, we expect graphs that would be constructed from primitives and objects detected in an image to feature missing nodes and a lot of false positive nodes. Due to the computational complexity, the practical applications of the parser have been shown to be constrained to graphs of about 30 nodes [61, 15].

3.1.2 Our approach

We propose to represent a shape prior by means of a graph grammar and potentials associated to arcs and nodes of graphs generated from a grammar. We propose to represent a facade segmentation in terms of a graph of facade parts and their positions. Finally, we propose a method for finding the optimal segmentations within the framework, where the inference of the structure of the model is separated from the inference of positions of parts for a given fixed structure. The benefit of this approach is that the less reliable randomized or greedy optimization algorithm is limited to structure inference, while a more effective technique is applied for optimizing positions of the parts. We also introduce a novel energy fusion scheme for combining object detections with texture classification. We evaluate experimentally the influence of each of the contributions on parsing performance.

We assume a factor graph-based model of a structure of a single facade. Models of this type have proven effective in applications like pedestrian detection and pose tracking [13, 11, 30, 71]. A factor graph is a bipartite graph with two types of nodes. The 'variable nodes' correspond to geometric primitives. The 'factor nodes' can correspond to constraints on the composition of the primitives, or to parts (basic objects) defined by a number of geometric primitives. An example factor graph is presented in section 3.2. We create a model of a facade encoding both the structure and positions of parts by transforming the factor graph into a graphical model: variable nodes are assigned variables defining positions of the geometric primitives in space. The factors are assigned potentials. Some of the potentials penalize violation of the constraints. Others evaluate hypotheses of the positions of parts against image cues. The potentials are defined in figure 3.2. The minimum energy labeling of the graphical model corresponds to positions of parts that best agree with image cues. The minimum energy is a measure of fitness of the graph to the image. We optimize over the positions of the parts with a state-of-the-art solver for calculating most likely configurations of graphical models.

For modeling variation of model structure, we propose a particular type of a Neighborhood Controlled Embedding (NCE) graph grammar. In other words, we use a grammar to represent a set of graphs, each corresponding to a different facade structure. The graph grammar paradigm is presented in section 3.3. An overview of

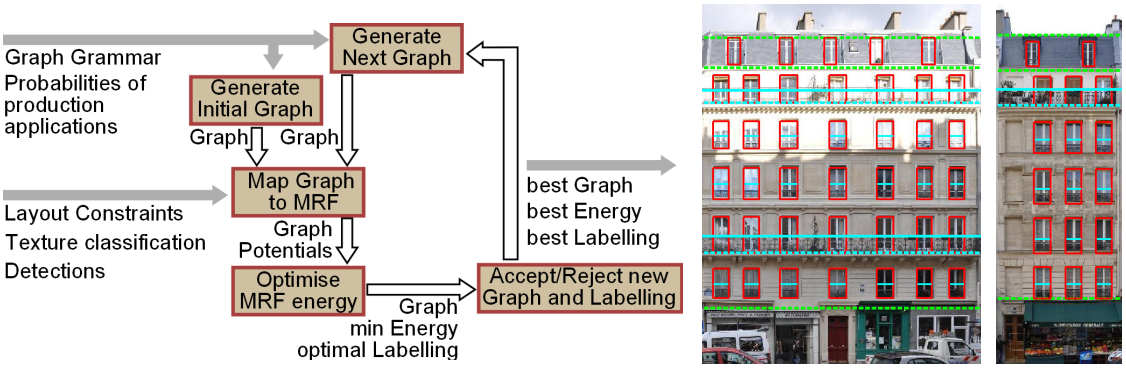


Figure 3.1: A diagram of our parsing algorithm *(left)*. Results of facade parsing using our approach *(right)*. The green snaplines separate the sky, roof, wall and shop classes. The cyan lines mark borders of running and single-window balconies.

the proposed inference algorithm is presented in figure 3.1. A single graph derived from the grammar can be encoded by its derivation tree. The exploration of the 'structural space' of derivation trees is performed by a sampling procedure. Given a current graph, our algorithm generates a new candidate graph by perturbing the sequence of grammar productions used to generate the former one. The perturbation consists in randomly re-generating a subtree of the derivation tree. This way both local and global structure changes are possible in a single step, depending on the choice of the root of the regenerated subtree. A local move is obtained by selecting a node close to the leaves of a parse tree, and a global move results from re-generating a deep subtree rooted close to the root of a parse tree. The candidate structure is accepted if the minimum energy of the corresponding graphical model is lower than the one of the current graph. The optimization procedure is detailed in section 3.6.

3.1.3 Contribution

The main novelties of the proposed method include:

1. the modeling framework where inference over part positions and inference of model structure are separated,
2. the application of a graph grammar to modeling variation of model structure and a method of exploring the structure space,
3. a late fusion scheme for combining texture classification and object detections.

Table 3.1: The most frequently used symbols in this chapter

G	a factor graph; defined on page 32
V	the set of variable-type vertexes of a factor graph; p. 32
F	the set of factors of a factor graph; p. 32
F^o	the set of factors $F^o \subset F$ representing objects; p. 32
F^c	the subset of factors $F^c \subset F$, representing constraints (on relative position and size of facade elements); p. 32
$\mathcal{E}$	the set of edges of a factor graph; p. 32
λ^{v}	a function assigning a label to each variable-type vertex in a factor graph; p. 32
λ^{f}	a function assigning a label to each factor in a factor graph; p. 32
$\mathcal{L}^{\mathsf{v}}$	the set of labels of variable-type vertexes of a factor graph; p. 32
$\mathcal{L}^{\mathsf{f}}$	the set of labels of factor-type vertexes of a factor graph; p. 32
$\mathcal{T}$	the set of object classes; p. 32
$\mathcal{G}$	a graph grammar; p. 33
S	a starting graph of a graph grammar, by convention it consists of a single node; p. 33
$\mathcal{L}^T$	a set of terminal variable node labels in a graph grammar; p. 33
$\mathcal{L}^N$	a set of nonterminal variable node labels in a graph grammar; p. 33
$\mathcal{L}^{\mathsf{e}}$	a set of factor labels in a graph grammar; p. 33
$\mathcal{P}$	a set of graph grammar productions; p. 33
D	a daughter graph in a graph grammar production; p. 33
$\mathcal{C}$	a set of connection instructions in a graph grammar production; p. 33
$n_{t,\rho}$	the number of times production ρ has been applied to graph node t in the training data; p. 39
$N_{t,\rho}$	the augmented count of times production ρ has been applied to graph node t in the training data; this number is always nonzero; p. 39
$\mathbf{x}$	the vector of positions of all geometric primitives in the graph; p. 40
$\mathbf{z}$	the vector resulting from concatenation of discrete variables encoding types of productions applied to generate a given graph; it is a parametrization of the graph structure; p. 43
$E(\mathbf{x}, \mathbf{z})$	total energy of a segmentation; p. 43
$E^{\text{struct}}(\mathbf{z})$	the energy corresponding to the structure of the graph; p. 37
$E^{\text{pos}}_{\mathbf{z}}(\mathbf{x})$	the energy component corresponding to the positions of facade elements for a given structure; p. 40
$\phi^o_f(\mathbf{x}_f)$	the potential assigned to factor $f \in F^o$ and evaluating the position $\mathbf{x}_f$ of the facade element corresponding to f; p. 40
$\phi^c_f(\mathbf{x}_f)$	the potential assigned to factor $f \in F^c$ and evaluating satisfaction of constraints by a relative position of a number of facade elements, encoded by $\mathbf{x}_f$; p. 40
$E^t(\mathbf{x}_f, l_f)$	the texture-related energy component; p. 40
$E^d(\mathbf{x}_f, l_f)$	the detection-related energy component; p. 40
$E^{\text{pix}}(I_p, l_f)$	the energy corresponding to labeling pixel I_p with the label l_f; p. 40

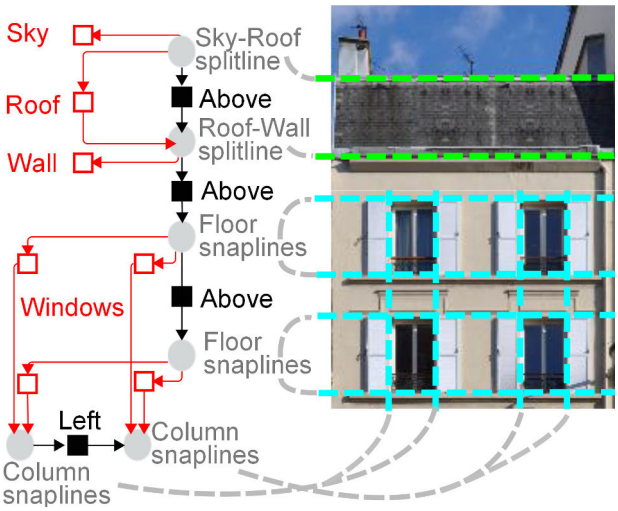


Figure 3.2: A simplified facade model and its interpretation. The gray disks are **variable nodes**, the rectangles represent factors: the black ones model **constraints** and the red ones represent **parts** of the model (for readability, the labels for individual window factors have been substituted with a single label '*Windows*'). The dashed lines in the image represent the **splitlines** (green) and the **snaplines** (cyan) encoded by variable nodes.

3.2 Facade model with fixed structure

We model parts of a facade and their spatial relations as a factor graph. A labeled factor graph $G = (V, F, \mathcal{E}, \lambda^{\mathsf{v}}, \lambda^{\mathsf{f}}, \mathcal{L}^{\mathsf{v}}, \mathcal{L}^{\mathsf{f}})$ consists of a set of variable nodes V, a set of factor nodes F, a set of directed edges $\mathcal{E} \subset (V \times F) \cup (F \times V)$ and functions $\lambda^{\mathsf{v}} : V \to \mathcal{L}^{\mathsf{v}}$, and $\lambda^{\mathsf{f}} : F \to \mathcal{L}^{\mathsf{f}}$, assigning labels $l_v \in \mathcal{L}^{\mathsf{v}}$, and $l_f \in \mathcal{L}^{\mathsf{f}}$ to the nodes and factors.

Each variable node $v \in \mathcal{V}$ is a variable x_v encoding the position of a geometric primitive. The set of factors $F = F^o \cup F^c$, where F^c and F^o are the sets of constraint- and object-type factors, and $F^c \cap F^o = \emptyset$. A constraint on a relative position of a pair of primitives is expressed by a constraint factor $f^c \in F^c$, connected to the pair of corresponding variable nodes. A part of the model, defined by several geometric primitives, is represented by an object factor $f^o \in F^o$, connected to the variable nodes encoding positions of the primitives. Labels assigned to object factors $\lambda^{\mathsf{f}}(f^o)$ correspond to classes of objects that can be represented by the factors. The set of object classes $\mathcal{T} \subset \mathcal{L}^{\mathsf{f}}$ consists of the labels of object factors and can be defined as $\mathcal{T} = \{\lambda^{\mathsf{f}}(f^o) | f^o \in F^o\}$.

Our approach, applied to facade modeling, is illustrated in figure 3.2. To easily accommodate alignment, non-overlap and adjacency of facade elements, we assume a line-based model. The variable nodes encode positions of splitlines and snaplines (nodes *floor* and *column* represent pairs of lines). The factor nodes correspond to objects (the set of object classes $\mathcal{T} = \{sky, roof, wall, window\}$) and constraints (*above* and *left*). The *sky* and *wall* factors are attached to a single splitline each because they extend from the splitline to image boundary.

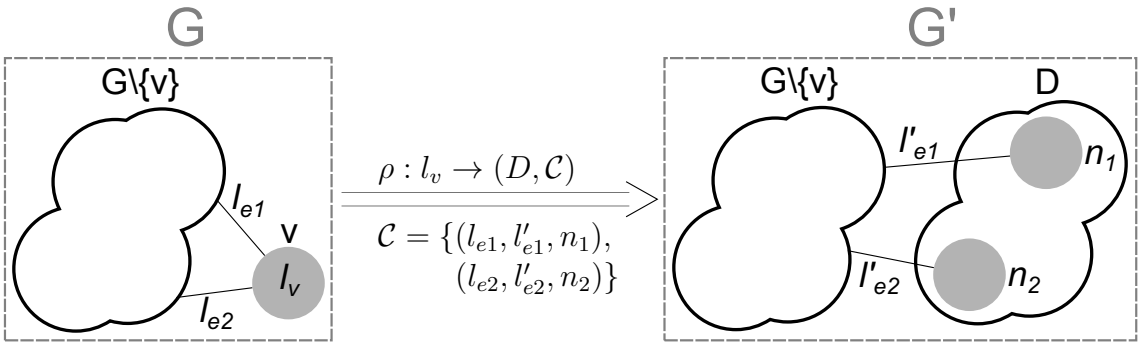


Figure 3.3: Application of a graph grammar production $\rho : l_v \to (D, \mathcal{C})$ to a graph G results in a graph G', in which a node v, labeled l_v, is substituted with a subgraph isomorphic to D. The subgraph is connected to the graph $G \setminus \{v\}$, resulting from removing the node v from G, according to the connection instructions $\mathcal{C}$.

3.3 Modeling structure variation with graph grammars

Our algorithm generates models of the type presented in section 3.2 using a graph grammar. We use the Neighborhood Controlled Embedding grammars of graphs with edge labels (edNCE grammars). Below we review the mechanism briefly. For detailed explanation, the reader is referred to the handbook by Rozenberg et al. [49], and to the work of Klempien-Hinrichs [23] for the hypergraph case. For consistency with section 3.2, we present the paradigm for factor graphs, i.e., we treat factors the way (hyper-) edges are treated in the literature.

A grammar $\mathcal{G} = (S, \mathcal{L}^T, \mathcal{L}^N, \mathcal{L}^{\mathsf{e}}, \mathcal{P})$ consists of a starting factor graph S, sets of terminal and nonterminal variable node labels $\mathcal{L}^T$ and $\mathcal{L}^N$, a set of factor labels $\mathcal{L}^{\mathsf{e}}$ and a set of productions $\mathcal{P}$. The starting graph S consists of a single nonterminal node. Productions are of the form $\rho : l_v \to (D, \mathcal{C})$, where l_v is a nonterminal node label, called 'mother node label', D is a labeled graph, called 'daughter graph', and $\mathcal{C}$ is a set of connection instructions. The production can be applied to a node v of a graph G if $\lambda^{\mathsf{v}}(v) = l_v$, where $\lambda^{\mathsf{v}}(v)$ is the label of node v. The process is illustrated conceptually in figure 3.3 and presented more formally in algorithm 1. First, a new graph D', isomorphic to D, is created and inserted into G as a disjoint subgraph. Then, the nodes of D' are connected to neighbors of v in G through factors created according to connection instructions $\mathcal{C}$. We use connection instructions of the form (l_e, l'_e, n), where l_e and l'_e are factor labels, and n identifies a node in D. For each $(l_e, l'_e, n) \in \mathcal{C}$, each factor connected to v and labeled l_e is copied into G. The copy is disconnected from v, reconnected to the node in D' identified by n, and relabeled to l'_e. Finally, node v is removed from G together with all its factors. An example graph grammar is presented in figure 3.4, and examples of the application of a production rule are shown in figure 3.5.

3.3.1 Deriving graphs from a grammar

A derivation develops a graph G by repeatedly applying productions to nonterminals in the graph. The process starts with the single node of the starting graph

Algorithm 1 Given a host graph $G = (V, F, \mathcal{E}, \lambda^{\mathsf{v}}, \lambda^{\mathsf{f}}, \mathcal{L}^{\mathsf{v}}, \mathcal{L}^{\mathsf{f}})$, apply production $\rho : l_v \to (D, \mathcal{C})$ to node $v \in V$, where $D = (V_D, F_D, \mathcal{E}_D, \lambda^{\mathsf{v}}_D, \lambda^{\mathsf{f}}_D, \mathcal{L}^{\mathsf{v}}, \mathcal{L}^{\mathsf{f}})$. For notational convenience we assume the sets V and V_D, and F and F_D, are pairwise disjoint.

Require: $\lambda^{\mathsf{v}}(v) = l_v$

procedure ApplyProduction(G,v,ρ)

▷ Insert D to G as a disjoint subgraph

$V \leftarrow V \cup V_D$
$F \leftarrow F \cup F_D$
$\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}_D$
$\lambda^{\mathsf{v}} \leftarrow \lambda^{\mathsf{v}} \cup \lambda^{\mathsf{v}}_D$
$\lambda^{\mathsf{f}} \leftarrow \lambda^{\mathsf{f}} \cup \lambda^{\mathsf{f}}_D$

▷ Connect D to $G \setminus v$ using connection instructions

for all $f \in F$ s.t. $((f, v) \in \mathcal{E}) \vee ((v, f) \in \mathcal{E})$ **do**
for all $(l, l', n) \in \mathcal{C}$ s.t. $\lambda^{\mathsf{f}}(f) = l$ **do**

▷ Copy factor f and relabel the copy to l'

$F \leftarrow F \cup \{f'\}$ ▷ create a new factor f'
$\lambda^{\mathsf{f}}(f') \leftarrow l'$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(f', v') | ((f, v') \in \mathcal{E}) \wedge (v' \neq v)\}$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(v', f') | ((v', f) \in \mathcal{E}) \wedge (v' \neq v)\}$
if $((f, v) \in \mathcal{E})$ **then**
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(f', n)\}$
else
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(n, f')\}$
end if
end for

▷ Remove factor f from the graph

$\mathcal{E} \leftarrow \mathcal{E} \setminus \{(v', f) | v' \in V\}$
$\mathcal{E} \leftarrow \mathcal{E} \setminus \{(f, v') | v' \in V\}$
$F \leftarrow F \setminus \{f\}$
end for

▷ Remove node v from the graph

$V \leftarrow V \setminus \{v\}$
end procedure

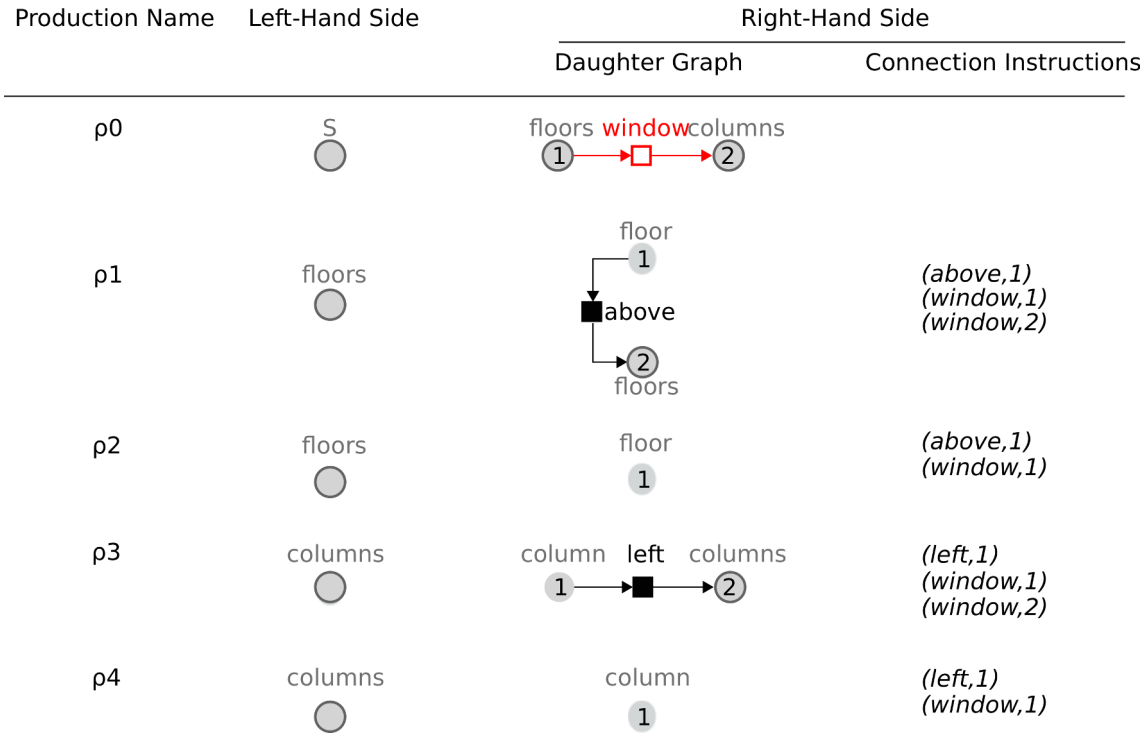


Figure 3.4: A toy edNCE grammar. Each row represents a single production. The node labeled S is a starting node of the grammar. The numbers assigned to nodes of the daughter graphs are used by the connection instructions (l_e, n), which say that an edge labeled l_e should be reconnected to a node n in the daughter graph of the production. In the presented grammar productions, we do not relabel edges, thus the short notation for connection instructions: (l_e, n) instead of (l_e, l'_e, n). The outlined disks represent nonterminal nodes; the other disks correspond to terminals, i.e., they cannot be rewritten. See caption of figure 3.2 for the interpretation of remaining symbols.

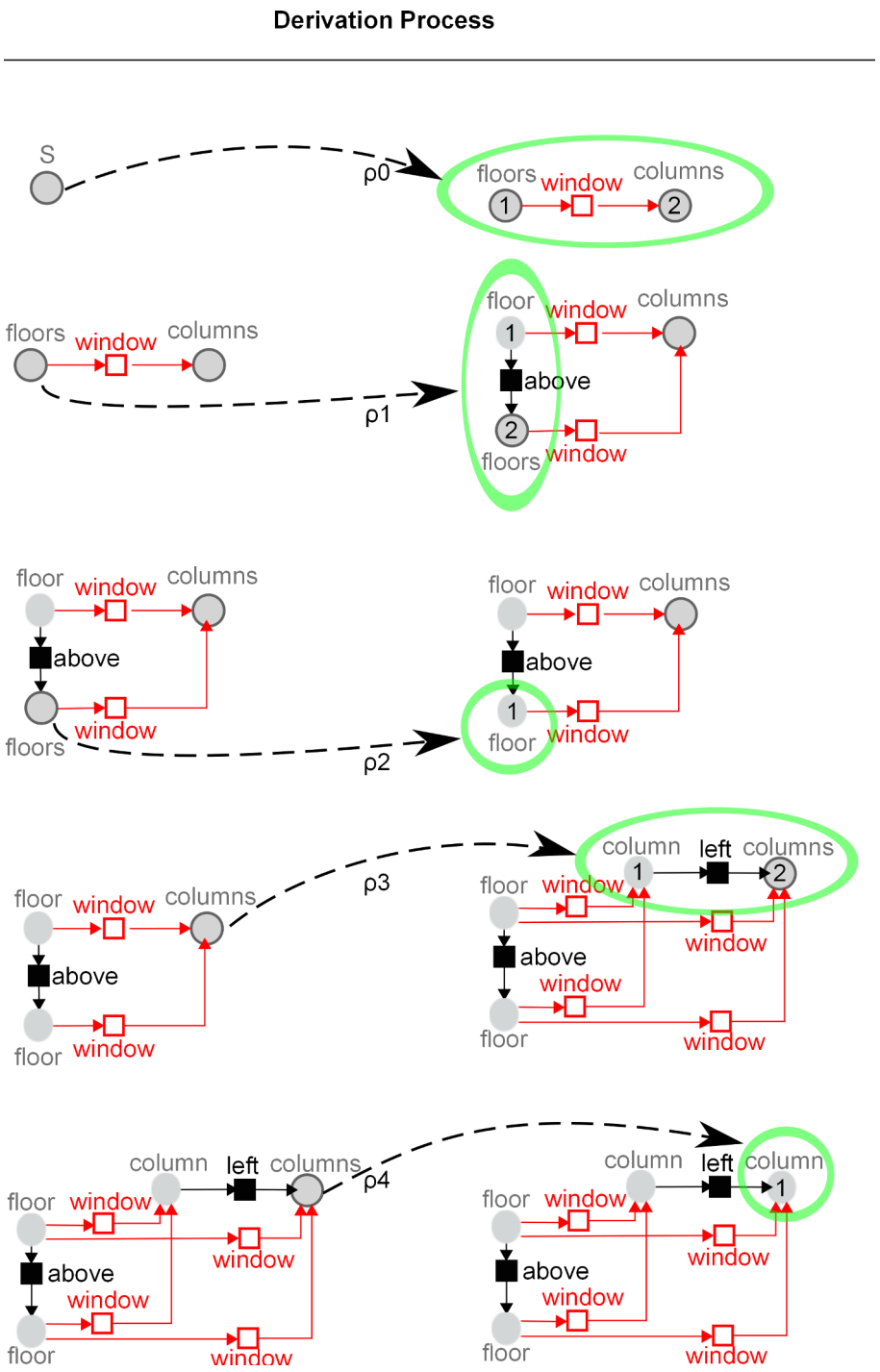


Figure 3.5: An application of the grammar presented in figure 3.4 to derive a graph. Each row presents the application of a single production. A dashed arrow connects a nonterminal node with a subgraph that replaces it.

$G = S$. Then, productions are applied recursively to nonterminal nodes in G, until it contains no more nodes with nonterminal labels. A sequence of production applications can be arranged into a derivation tree, in which each node corresponds to an application of a grammar production to a particular nonterminal node of the graph G. Therefore, each node of a derivation tree represents a pair of a node of graph G, and a production applied to it. In particular, the root corresponds to the nonterminal node constituting the starting graph S, and the production applied to it. An edge from a parent to a child node in a derivation tree means that the child represents a node of graph G inserted by the production encoded by the parent. An example derivation from the graph grammar presented in figure 3.4 is visualized in figure 3.5 and the corresponding derivation tree is shown in figure 3.6.

3.3.2 Energy of a derivation

We denote the derivation tree by T and its nodes by t. The production associated to node t is denoted ρ_t. To identify a graph derived from the grammar, we denote a vector resulting from concatenating discrete variables encoding all the productions ρ_t, used to derive the graph, by $\mathbf{z}$ and define the energy of the derivation tree $E^{\text{struct}}(\mathbf{z})$ as the sum of energies of individual production applications. We emphasize that in assigning energies to individual production applications, we treat differently productions applied to different nonterminal nodes with the same label. This matters, for example, when the grammar contains a recursion that inserts a sequence of nodes. Depending on the likely number of elements in the sequence, the penalty for terminating the recursion will depend on the depth of the recursion, even though each time the recursive production is applied to a node with the same label. In the facade modeling example from figure 3.4, we have a recursive production ρ_3, which inserts nodes of class *column*, and production ρ_4, which terminates the recursion. Both of these productions can be applied to a node of nonterminal class *columns*. However, the preference for one of the productions over the other may depend on the number of *column* nodes already inserted. For example, the two *columns* nodes in figure 3.5 have different interpretation: the first node corresponds to the first window column, the second node to the second column. In an extreme case, if all the buildings in our dataset had two window columns, we would almost always apply ρ_3 to the first node and almost always select ρ_4 to develop the second node, even though they have the same label. A node t of the derivation tree unambiguously identifies the substituted nonterminal node. We therefore define the energy of a derivation in terms of the derivation tree

$$E^{\text{struct}}(\mathbf{z}) = \sum_{t \in T} -\log p_t(\rho_t) \ , \tag{3.1}$$

where $p_t(\rho_t)$ is the likelihood of choosing production ρ_t out of all alternatives applicable to the nonterminal node identified by t. The likelihoods can be estimated from ground truth parse trees associated to training data.

The data structure used for estimating and storing the likelihoods p_t is called an And-Or Tree (AOT). We briefly describe the structure here. The reader is referred to [73] for a more detailed explanation. The tree consists of two types of nodes, the 'or' nodes, which in our case correspond to nonterminal nodes of the derived graph,

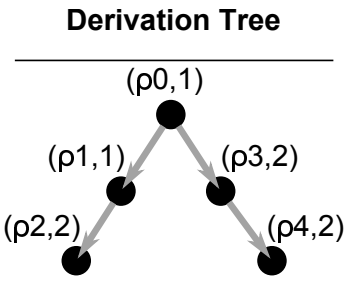


Figure 3.6: A derivation tree, reflecting the interdependence of production applications presented in figure 3.5. Each node corresponds to application of a production, represented by a pair (ρ, n), where ρ is the production name and n is the index of the substituted node, as defined in the parent production (see figure 3.4).

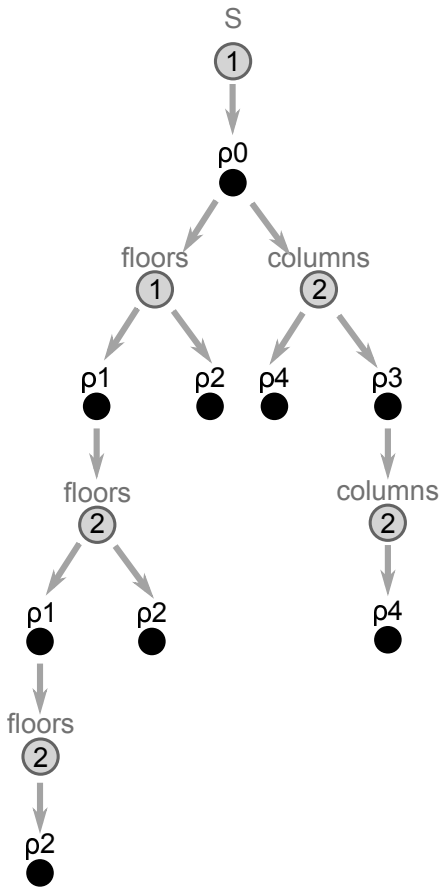


Figure 3.7: An And-Or Tree corresponding to the grammar presented in figure 3.4. The gray, circumscribed circles correspond to 'or' nodes, representing nonterminal nodes of the derived graph. The name 'or node' emphasizes the fact that each nonterminal can be substituted using one of possibly many graph grammar productions. The numbers inside the 'or' nodes are their indexes defined in the productions that inserted them (see figure 3.4). Each black circle corresponds to an 'and' node, representing a possible production application. The depicted tree represents all possible derivation that represent facades with at most three floors and two columns. However, the grammar enables arbitrarily large numbers of floors and columns. See the section 3.3.2 for the details on how the energies associated to alternative productions are evaluated if some of the alternatives are not represented in the AOT.

and the 'and' nodes, corresponding to production applications. Edges are enabled only between nodes of different types. A parent-child edge from an 'or' node to an 'and' node means that the graph node represented by the 'or' node of the AOT can be substituted by a production represented by the 'and' node. A link from an 'and' parent to a an 'or' child means that the production corresponding to the 'and' node inserts into the graph a nonterminal node represented by the 'or' node of the AOT. An example AOT is presented in figure figure 3.7.

Note, that each of the 'or' nodes represents a particular node that can appear in some derivations and not in some others. Therefore, an 'and' node represents an application of a production to a particular node, and not the production itself. A derivation tree can be obtained by selecting a single child of each 'or' node of an AOT and removing the subtrees rooted at the remaining children. In this sense, an AOT is a shared representation of a number of derivation trees.

We attribute to each 'and' node the number of the training examples that contain the corresponding production application. We use the frequentist approach to estimate the probabilities $p_t(\rho_t)$. We make the estimation more robust to cases not seen in the training set by adding a small number ϵ to the observed numbers of production applications. We denote the number of times a production ρ_t has been applied to the node identified by t in the training set by n_{t,ρ_t}, we define

$$N_{t,\rho_t} = n_{t,\rho_t} + \epsilon \ . \tag{3.2}$$

By $\mathcal{P}(t)$ we denote the set of productions that can be applied to the node identified by t. The probability of applying a particular production ρ_t to a particular node t becomes

$$p_t(\rho_t) = \frac{N_{t,\rho_t}}{\sum_{\rho \in \mathcal{P}(t)} N_{t,\rho}}. \tag{3.3}$$

During training, the And-Or Tree is created progressively. First the root 'or' node is inserted corresponding to the single-node starting graph. Then, each ground truth parse tree is used to update the tree. A parse tree is traversed from root to leaves. Each node of the parse tree corresponds to a production application and, equivalently, to an 'and' node in the AOT. If the corresponding node in the AOT exists, the corresponding count n_{t,ρ_t} is incremented. If it does not exist, it is created with a count equal to one. In consequence, the AOT that we construct does not contain all the possible production applications. Otherwise, for grammars with recursion, this would mean an infinite tree depth. However, during inference, the algorithm can create graphs that it has not seen during training, due to the robust estimation of $p_t(\rho_t)$, assigning a small probability to unseen configurations. If a branch of the derivation tree, that is not represented in the AOT, is constructed during inference, all candidate productions $\rho \in \mathcal{P}(t)$ applicable to a given node t get the count $N_{t,\rho} = \epsilon$. Therefore, they are equally probable according to (3.3), which reflects the lack of prior knowledge on their likelihood.

3.4 Fitness energy

A factor graph derived from a grammar defines the number and type of objects present in the scene and the relations between them. The precise positions of the

objects and geometric primitives are defined by variables x_v associated with the nodes of the model. We denote a concatenation of all the parameters x_v by $\mathbf{x}$. In order to evaluate how well the model explains the input image I, we define an energy function $E^{\text{pos}}(\mathbf{x})$, evaluating positions of the parts.

The energy is defined as a sum of potentials assigned to the factors of the graph, forming a graphical model. To each factor f we assign a potential $\phi_f(\mathbf{x}_f)$ defined in terms of a vector of variables $\mathbf{x}_f = (x_v)_{(v,f)\in\mathcal{E}\vee(f,v)\in\mathcal{E}}$ corresponding to neighboring nodes. Different types of factors are assigned different potentials. Constraint-type factors $f \in F^c$ receive potentials ϕ^c_f that penalize violation of predefined constraints on relative position of geometric primitives. Each object-type factor $f \in F^o$ receives a potential ϕ^o_f evaluating an object hypothesis. The class of the hypothesized object is encoded by the label of the factor $\lambda^{\text{f}}(f)$ and its position is defined by a bounding box, which consists of the geometric primitives represented by the nodes to which the factor is connected. For example, in the model in figure 3.2, a *window* factor evaluates a hypothesis defined by a column and a floor node, each of which encodes a pair of snaplines. The position component of energy is the sum of all these potentials:

$$E^{\text{pos}}(\mathbf{x}) = \sum_{f\in F^o}\phi^o_f(\mathbf{x}_f) + \sum_{f\in F^c}\phi^c_f(\mathbf{x}_f)\ . \tag{3.4}$$

3.4.1 Object-type potentials

We use two kinds of bottom-up cues for measuring the fitness of the model to an input image: texture classification and object detections. Texture classification is given in the form of an energy value $E^{\text{pix}}(I_p, \tau)$ for each point p belonging to image I and a fixed-size image patch I_p, centered at p, and for each class of parts $\tau \in \mathcal{T}$. The energy $E^{\text{pix}}(I_p, \tau)$ measures the log-likelihood of observing a patch I_p if the pixel represents an object of class τ. Texture classification is available for each class of the parts. However, detections might be available for a subset of the classes $\mathcal{T}^{\text{det}} \subset \mathcal{T}$ only. They have the form of bounding boxes y_i with weights w_i, representing the confidence of the detections. Recall that a class of an object represented by a factor f is encoded in its label l_f. Object factors are expressed in terms of texture and detection-based energies E^t and E^d as:

$$\phi^o_f(\mathbf{x}_f) = \begin{cases} \alpha^t E^t(\mathbf{x}_f, l_f) + \alpha^d E^d(\mathbf{x}_f, l_f) & \text{if } l_f \in \mathcal{T}^{\text{det}} \\ E^t(\mathbf{x}_f, l_f) & \text{otherwise,} \end{cases} \tag{3.5}$$

where α^t and α^d are constant coefficients. We call the above scheme 'late fusion', because it uses detections directly in the energy formulation, contrary to the 'early fusion' scheme [59, 44, 47, 33], where detections are projected down to pixel level and energy is evaluated on individual pixels only (note that the scheme presented in appendix A.1.1 is also an instance of early fusion). In the latter scheme, an energy assigned to each pixel is a function of the sum of confidence scores of all detections that overlap the pixel. Usually detectors return a large number of overlapping candidate bounding boxes. In this case early fusion results in 'blurring' object boundaries in the optimal segmentations. The effect is presented in figure 3.8. On the other hand, if aggressive non-maximum suppression is used, useful information

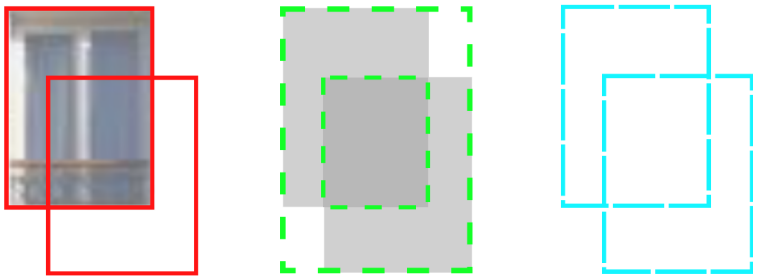


Figure 3.8: The advantage of late fusion against early fusion. *Left*: two detections, a true positive and an inaccurate detection (red boxes), a common result of object detection. *Middle*: a confidence map produced by early fusion and possible optimal object hypotheses, depending on parameters (green boxes). Note that none of the hypotheses corresponds to the ground truth bounding box, even though one of the detections matched it perfectly. We call this effect 'blurring' of object hypotheses. *Right*: possible optimal object hypotheses for late fusion and our detection-based potentials. One of the hypotheses corresponds to the ground truth bounding box.

may be lost. Non-maximum suppression retains a detection if there are no stronger detections within a neighborhood of a given size, and discards it otherwise. Such operation is local and does not take into account the long-distance alignments, important in man made objects. It can yield misaligned detections, even if aligned candidate bounding boxes exist, if there are stronger noisy detections within their neighborhoods. The proposed late fusion scheme, together with the detection-based potential presented below, prevents blurring object boundaries and enables taking advantage of the full set of detection candidates. An experimental comparison of the late and the early fusion is presented in section 3.7.

Estimating the coefficients α^t and α^d requires knowledge of the true model structures and part positions for images of the training set, and can be formulated as max-margin Markov Random Field training (see, e.g., [25]).

Detection-based Potentials. We need the energy function to be robust to missing detections and false positives, and thus to rank object hypotheses based on bounding boxes of individual detections to avoid the effects presented in figure 3.8. Given a function $d(\mathbf{x}_f, y_i)$, evaluating a distance between an object bounding box defined by vector $\mathbf{x}_f$ and a detection y_i of weight w_i, the detection-based potential is expressed by the distance to the 'best' detection in its neighborhood, $\min_i(d(\mathbf{x}_f, y_i) - \beta^w w_i)$, for some constant β^w, or by a constant penalty β^d if there are no nearby detections. A possible interpretation of β^d is the distance after which the detection does not influence the potential. The term $\beta^w w_i$ increases the radius of influence of detections depending on weights w_i. More formally, the detection-based potentials take the form:

$$E^d(\mathbf{x}_f, l_f) = \min\{\min_{i \in \mathrm{Det}(l_f)} (d(\mathbf{x}_f, y_i) - \beta^w w_i), \beta^d\} \,, \tag{3.6}$$

where $\mathrm{Det}(l_f)$ is a set of indexes of detections of class l_f. We let the potentials take negative values for good object hypotheses, so that when comparing models with

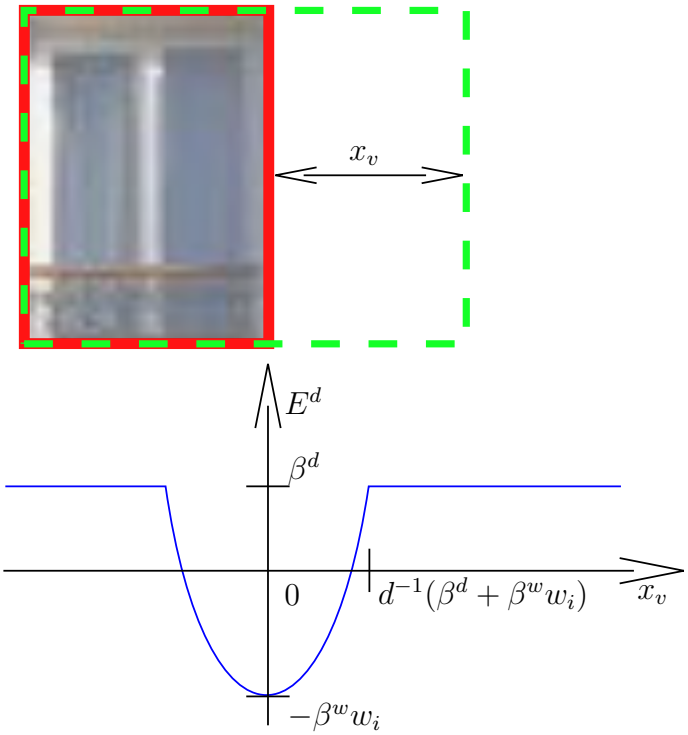


Figure 3.9: The interpretation of the detection-based potential. A detection is shown as a red rectangle, the evaluated hypothesis is shown as green dashed rectangle. The detection energy is shown below as a function of the position of one side of the hypothesis, with the other three sides fixed and aligned with the detection. We abuse the notation by using $d^{-1}()$ to denote x_v, the coordinate of the misaligned side of the hypothesis bounding box with respect to the corresponding side of the detection.

two different structures the model with missing hypotheses gets a higher energy. The interpretation of the distance function is illustrated in figure 3.9. In our experiments we use the squared Euclidean distance as distance function d. We currently adjust the parameters β^w and β^d manually.

Texture-based Potentials. We define the texture-based energies E^t as in [60]. For a factor f the energy is evaluated over the corresponding rectangular region in the image, denoted $B(\mathbf{x}_f)$:

$$E^t(\mathbf{x}_f, l_f) = \sum_{p \in B(\mathbf{x}_f)} E^{\text{pix}}(I_p, l_f) \; . \tag{3.7}$$

where l_f is the factor label, encoding its class.

Objects positioned on the background of other objects, like windows on a wall in figure 3.2, are treated specially. We denote the set classes of such objects as $\mathcal{T}^{\text{fg}} \subset \mathcal{T}$. We denote the class of the background object associated with factor label $l_f \in \mathcal{T}^{\text{fg}}$ as $c^{\text{bg}}(l_f)$. For each label $l_f \in \mathcal{T}^{\text{fg}}$, $c^{\text{bg}}(l_f)$ is defined uniquely, for example, the attic windows (placed over the background of the roof) are represented by a different class than the facade windows (on wall background). The potential of the background object is evaluated over its whole bounding box (including the regions

of the foreground object) and the potential of the foreground object becomes:

$$E^t(\mathbf{x}_f, l_f) = \sum_{p \in B(\mathbf{x}_f)} E^{\text{pix}}(I_p, l_f) - E^{\text{pix}}(I_p, c^{\text{bg}}(l_f)) \ , \tag{3.8}$$

so that the sums of background potentials over the foreground region cancel out when the potentials are added together.

3.4.2 Composition potentials

In general, the potentials ϕ^c that express priors on composition of parts can take arbitrary forms. We use the potentials to enforce a set of predefined constraints $g_f(\mathbf{x}_f)$, by penalizing values of vector $\mathbf{x}_f$ that violate the constraints:

$$\phi^c_f(\mathbf{x}_f) = \begin{cases} 0 & \text{if} \quad g_f(\mathbf{x}_f) = \text{true} \ , \\ \infty & \text{otherwise} \ . \end{cases} \tag{3.9}$$

The particular types of constraints that we use in our experiments are presented in section 3.7.

3.5 Total energy

The total energy of a model is a combination of the derivation energy, evaluating the structure of the model, and the fitness energy, evaluating the positions of parts:

$$E(\mathbf{x}, \mathbf{z}) = \alpha^{\text{struct}} E^{\text{struct}}(\mathbf{z}) + \alpha^{\text{pos}} E^{\text{pos}}_{\mathbf{z}}(\mathbf{x}) \ , \tag{3.10}$$

where $\mathbf{x}$ denotes the vector of positions of all geometric primitives and $\mathbf{z}$ denotes the vector resulting from concatenation of discrete variables encoding the types of applied productions. We denote the position energy $E^{\text{pos}}_{\mathbf{z}}$ with the subscript $\mathbf{z}$ to emphasize that it is defined for a given structure.

Given a set of true and perturbed facade structures $\mathbf{z}_i$ for ground truth images, we can determine optimal positions $\hat{\mathbf{x}}_i$ for each of the structures and use the optimal energy values $E^{\text{struct}}(\mathbf{z}_i)$ and $E^{\text{pos}}_{\mathbf{z}_i}(\hat{\mathbf{x}}_i)$ for estimating the coefficients α^{struct} and α^{pos} by max-margin training.

3.6 Optimization

The key to our approach is to employ different methods for minimizing the energy over the spaces of possible structures $\mathbf{z}$ and part positions $\mathbf{x}$. Optimization over the structure is inherently an ill-posed problem and requires a randomized [59, 52] or greedy [18, 60] exploration strategy. On the other hand, although inferring the optimal $\mathbf{x}$ for a given factor graph is known to be NP-hard, there exists a number of approaches experimentally proven to perform well in this task (for example [24, 26, 57]). Therefore, we propose to limit the application of the algorithm proposed by Teboul et al. [60] to structure exploration and use a state-of-the-art solver [24] to optimize over $\mathbf{x}$ for a fixed $\mathbf{z}$.

Given a derivation tree $\mathbf{z}^j$, the algorithm of [60] generates a candidate tree $\mathbf{z}^{\text{cand}}$, and sets

$$\mathbf{z}^{j+1} \leftarrow \begin{cases} \mathbf{z}^{\text{cand}} & \text{if} \quad \min_{\mathbf{x}} E(\mathbf{x}, \mathbf{z}^{\text{cand}}) < \min_{\mathbf{x}} E(\mathbf{x}, \mathbf{z}^j) \ , \\ \mathbf{z}^j & \text{otherwise.} \end{cases} \tag{3.11}$$

This greedy technique differs from a Metropolis-Hastings random walk (see, e.g., [8]) in that it never accepts a candidate of lower energy. To generate $\mathbf{z}^{j+1}$ given $\mathbf{z}^j$, we randomly select a production application in the derivation tree and resample the subtree rooted at the selected production application. This enables both global and local changes to the structure, e.g., altering the number of columns without changing the number of floors. With general edNCE graph grammars, the derived graph can depend on the order in which disjoint subtrees of the derivation tree are developed. A grammar for which the yielded graph does not depend on the order of applying productions is called 'confluent'. We limit the scope of applications of the algorithm to confluent grammars, in order to avoid the dependence on the order of derivations when resampling a graph from a grammar. We consider that the grammars that are interesting from the point of view of facade modeling possess this property. In particular, the grammars used in our experiments are confluent.

Optimization over positions of the parts $\mathbf{x}$ is performed by means of the TRW-S algorithm [24]. It solves the dual to a linear relaxation of the problem of finding the minimum energy configuration of an undirected graphical model. We exploit the fact that the dual objective is a lower bound on minimum primal energy by stopping the optimization as soon as it exceeds the value of the best energy attained so far.

3.7 Experiments

We evaluate our algorithm in the task of parsing rectified photographs of building facades. In a number of experiments, we prove the advantage of separating the structure inference from the inference of positions of parts, compare the performance of our algorithm to that of a state-of-the-art parser, and compare the performance of late and early fusion. In the experiments, we use a Haussmannian facade grammar described in the following subsection.

3.7.1 The grammar of Hausmannian facades

The grammar we use for encoding the structure of Haussmannian facades is presented in figure 3.10. It encodes three different variations of window layout: i) there are no attic windows, ii) the attic windows are aligned with the windows on the facade, iii) the attic windows are not aligned with the windows on the facade. It models an unknown number of floors and window columns in the facade and the attic. The grammar is confluent and enables 'local' derivation of different parts of the graph. In particular, the number of floors, the number of columns and the type of attic are selected independently. This enables local changes to the structure of the graph during optimization.

An example model of a Haussmannian facade, generated from our grammar, is presented in figure 3.11. Each model generated from the grammar contains a variable

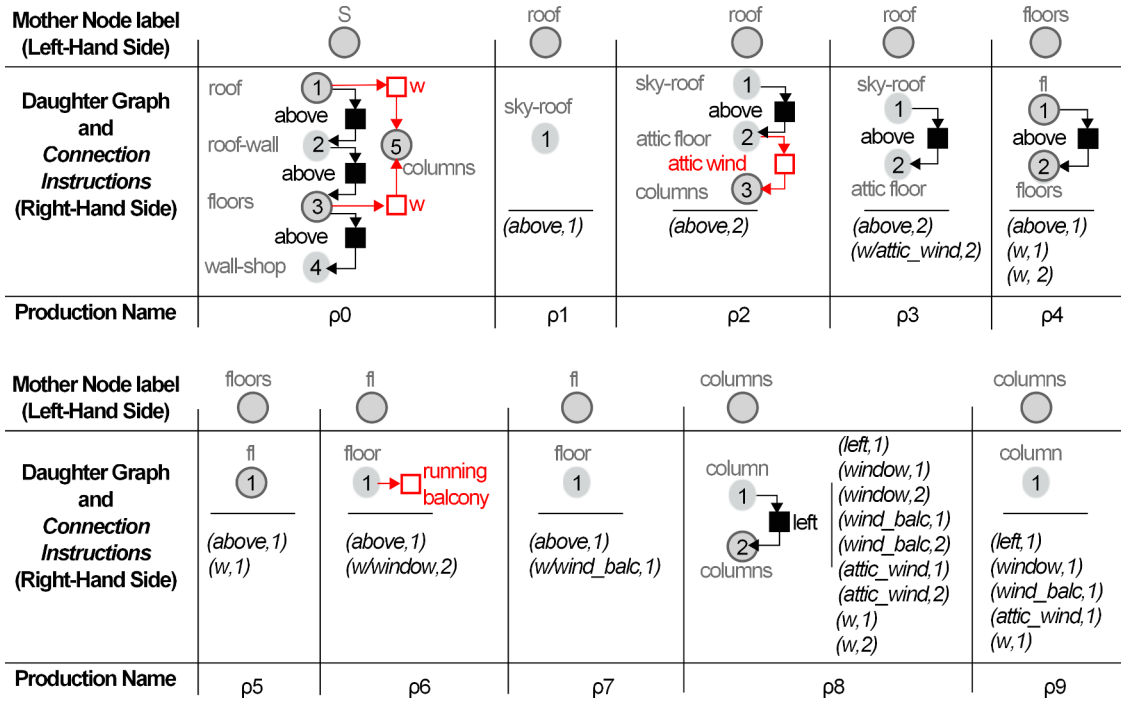


Figure 3.10: The grammar of Haussmannian facades. S is the start symbol of the grammar. Productions ρ_1, ρ_2 and ρ_3 express different patterns of attic window alignment. Productions ρ_4, ρ_5, ρ_6 and ρ_7 recursively encode a variable number of floors with two possible types of balconies: a running balcony and a balcony limited to a single window. Productions ρ_8 and ρ_9 encode a variable number of window columns. Factors labeled w can be interpreted as 'general' windows – they are specialized to *window* or *wind_balc* factors by productions ρ_6 and ρ_7. Connection instructions that do not relabel edges are presented in the short form (l_e, n). See the captions of figure 3.2 and 3.4 for an explanation of graphical symbols.

node encoding the position of each of the following geometric primitives: a *sky-roof* line (x_{sr}), a *roof-wall* line (x_{rw}) and a *wall-shop* line (x_{ws}). Additionally, the models contain a certain number of *floor* nodes encoding the positions of the window top, window bottom and balcony top snaplines (x_t, x_b, x_{tb}), and *column* nodes encoding the positions of the window left and window right snaplines (x_l, x_r). Parts of the facade are encoded as object factors, defined in terms of the geometric primitives. For example, a *window* factor encodes a window on a floor with a running balcony and a *wind_balc* factor encodes a window with a small balcony limited to the window cavity.

The constraints for the relative positions of the geometric primitives are expressed by potentials of constraint-type factors (Equation 3.9). They are summarized in table 3.2. The constants in these architectural constraints represent ranges for relative sizes of basic parts. The potentials penalize segmentations which clearly violate our model of a 'valid' facade. For instance, the constraint on the maximum gap between two consecutive floors prevents having a floor missing due to an unusual appearance or occlusions. The method is insensitive to moderate changes of the values. We set them manually on the basis of relative sizes of ground truth objects in the training images, although they could easily have been estimated automatically.

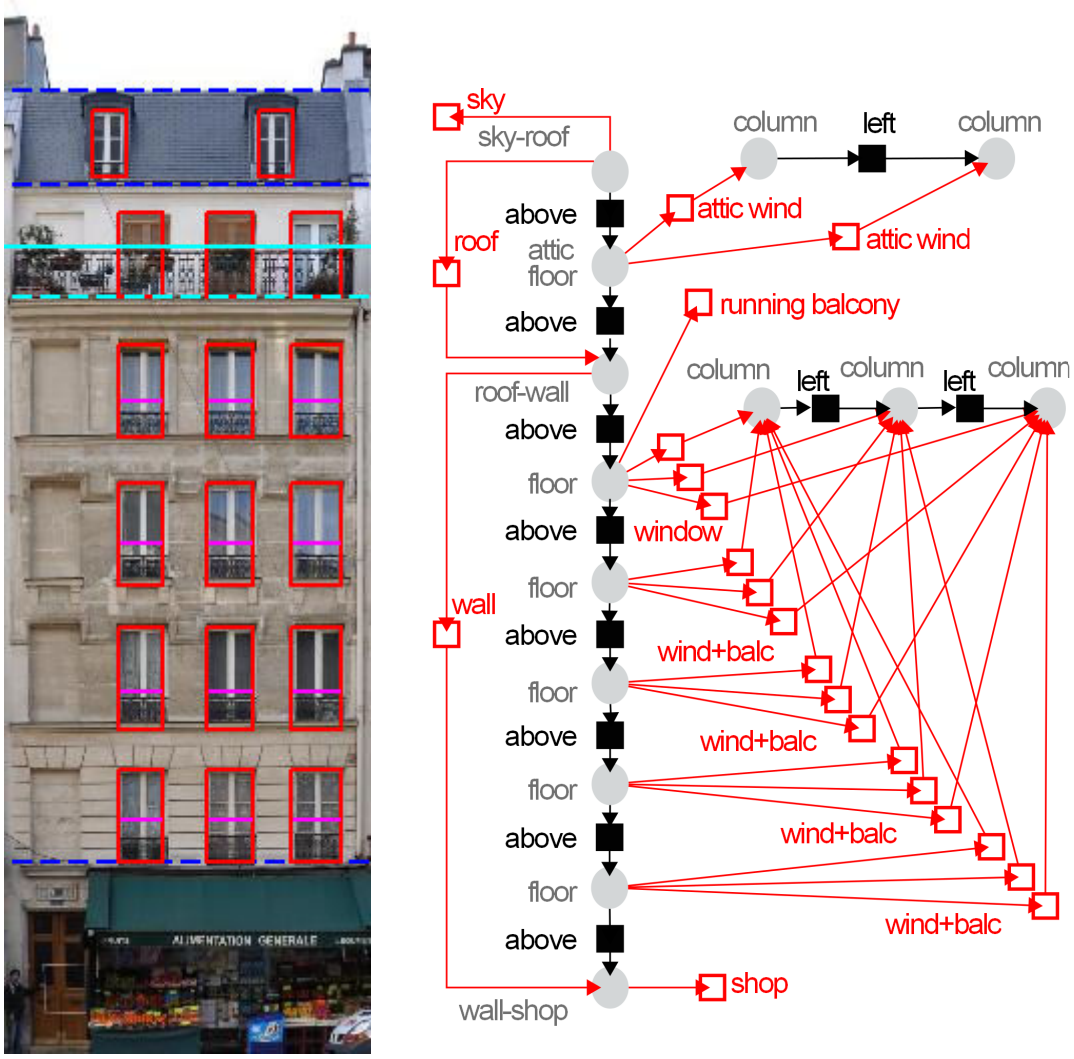


Figure 3.11: A facade segmentation and a graph representing its structure. See the captions of figure 3.2 and 3.4 for an explanation of graphical symbols.

3.7.2 Experimental settings

We run a Matlab implementation of the algorithm, with an external implementation of TRW-S solver by Kolmogorov [24], written in C, on an office computer (Intel Core i7 processor, 2.66 GHz). To keep the running time reasonable, we only consider a subset of the possible positions for splitlines and snaplines: we consider positions distributed evenly every four pixels along both dimensions when the detections are not used, and in experiments where detections are used we use the sides of their bounding boxes as candidate window split- and snapline positions. In our experiments, the average running time of the optimization algorithm was 3.5 minutes per image.

In each experiment we apply the same train-test scheme as in [33], with 80 training images and 20 testing images, and repeat the validation 5 times. We use the ECP dataset [59], containing 104 rectified images of Haussmannian facades. The original annotations provided with the dataset tend to be inaccurate, as they were generated from the grammar used in the paper (personal communication with the

Table 3.2: Constraints assigned to the constraint factors. Factors are defined by the labels of nodes to which they are connected and their own edge labels: $node\_label_1 \xrightarrow{factor\_label} node\_label_2$. See the text in section 3.7.1 for an explanation of the variable names.

Factor type	Constraints
sky-roof $\xrightarrow{above}$ *floor*	$x_{sr} \leq x_t$
floor $\xrightarrow{above}$ *roof-wall*	$x_t \leq x_{rw}$
roof-wall $\xrightarrow{above}$ *floor*	$x_t - x_{rw} \leq 0.7\,(x_b - x_t)$ $x_t - x_{rw} \geq 0$
floor $\xrightarrow{above}$ *floor'*	$(x'_b - x'_t) \leq 1.4\,(x_b - x_t)$ $(x_b - x_t) \leq 1.4\,(x'_b - x'_t)$ $(x'_t - x_b) \geq 0.1\,(x'_b - x'_t)$
floor $\xrightarrow{above}$ *wall-shop*	$(x_{ws} - x_b) \leq 0.5\,(x_b - x_t)$ $x_{ws} \geq x_b$
column $\xrightarrow{left}$ *column'*	$(x'_r - x'_l) \leq 1.4\,(x_r - x_l)$ $(x_r - x_l) \leq 1.4\,(x'_r - x'_l)$ $(x'_l - x_r) \leq 6.0\,(x_r - x_l)$ $(x'_l - x_r) \geq 0.5\,(x_r - x_l)$

authors). For example, in the annotations, attic windows are always aligned with the windows on the facade. But in fact the attic windows and facade windows in the images can be misaligned, due to architectural variations, or because both kinds of windows happen to be in different planes and the images are rectified. Similarly, in the annotations, the balconies either run along whole width of a floor or they are aligned with boundaries of a single window, whereas balconies visible in the images can be wider than an individual window, but not necessarily run through the whole width of a facade. Other annotations were prepared for the same images by the authors of [33]. They are more accurate, but do not respect structural constraints. For example, the windows in the same floor are not necessarily aligned, balconies can be misaligned with the corresponding windows, and pieces of doors can be floating above the ground. In the experiments presented in this section we use both ground truths, and the choice of a ground truth for each comparison is determined by the goal of the experiment.

3.7.3 Separate structure and position inference

To demonstrate the advantage of separating structure and position inference, we compare our algorithm against the method proposed by Teboul et al. [59, 58]. The comparison is made on the dataset presented with their paper and against their ground truth. The grammar presented in the paper models only limited variations of the facade structure: windows are arranged in a grid pattern, with roof and facade windows necessarily aligned. The structural variation is limited to the number of floors and columns, and to the choice of one of two types of balconies in each floor. For a fair comparison with this work, we introduce the same constraints to our grammar. The algorithm of [59] is based on texture classification only, so in one of the experiments we restrict our energy to texture-related terms. To define the

potentials, we use a random forest classifier [7] for texture classification, trained as in [60]. We also perform an experiment with the detection term, where windows are detected with the Viola-Jones classifier [62]. We use squared Euclidean distance as the distance function $d(\cdot)$ in the detection-based potentials defined in equation (3.6). By experimenting with several values, we set $\beta^d = 16$ and $\beta^w = 48$. According to (3.6), this means that poor detections with weights $w_i \approx 0$ would have a radius of influence equal to 4 pixels in the four-dimensional space of bounding boxes and the best detections with $w_i \approx 1$ influence the potential up to a distance of 8 pixels.

We run both methods using exactly the same per-pixel potentials. The algorithms are run for 5 minutes per image. The algorithm of Teboul et al. is re-run 10 times (instead of 5 times as reported in [58]) for 30 seconds and the best result is kept. The results are reported in table 3.3. As shown in the table, for 'easy' objects like shop, sky and wall, the position of which is determined by just a pair of split-lines, the performance is similar. The advantage of the more reliable optimization algorithm shows for the classes with a higher number of interactions, like windows and balconies. When using detections the accuracy increases even more, especially for the roof and balcony classes. The increase for the roof is related to the fact that the presence of roof windows enforces their background to be labeled as roof. Without the detections roof is often mistaken for sky. The position of balconies is tied to windows, and their detections improve accuracy for balconies as well.

Table 3.3: Comparison of results of the proposed parsing algorithm to the results of split grammar parsing proposed by Teboul et al. [59]. Rows of the matrix contain class-specific accuracies, i.e., the diagonal entries of confusion matrices. The first column presents results of split grammar parsing [59] with per pixel potentials obtained using a random forest classifier. The second column contains results of our graph grammar parsing, with the same potentials, that is, without detections (GG*). The third column contains results of the proposed algorithm, using the same per-pixel potentials as well as window detections (GG). As we do not model doors in our grammar, the door class has been merged with 'shop', to form a single 'shop/door' class.

	[59]	GG*	GG
roof	64	60	**69**
shop/door	**98**	**98**	**98**
balcony	58	66	**73**
sky	87	**91**	**91**
window	65	**75**	74
wall	83	86	**88**
total accuracy	78.9	82.6	**84.9**

3.7.4 Late vs. early fusion

We also compare the performance of early and late fusion for combining window detection and texture classification. We assume a general early fusion scheme where a new energy term is defined for each pixel. It is a convex combination of the per-pixel texture and detection energies: $E'_t = E^{\text{pix}}(I_p|c) + E_{\text{det}}(w_p|c)$. The per-pixel detection-based energy E_{det} depends on w_p, the cumulated weight of window detections overlapping pixel p. We set $E_{\text{det}}(w_p|c) = -\log p(w_p|c)$ and estimate the probabilities from the training data. We use the same detections and per-pixel

potentials as in comparison with split grammars in section 3.7.3. We use the full Haussmannian grammar, presented in section 3.7.1 and evaluate the result against the ground truth of Martinović et al. [33]. As presented in the last segment of table 3.4, parsing results for window and balcony are notably better with late fusion.

Table 3.4: Performance of our parsing algorithm with early fusion (left) and late fusion (right) potentials. Random forest-based texture classification was used to estimate the per-pixel potentials. We run the experiment using the full Haussmannian grammar, and evaluate results with respect to the ground truth produced by Martinović et al. [33]. The different grammar and ground truth result in different accuracies when compared to table 3.3.

	early fusion	late fusion
roof	79	80
shop/door	85	85
balcony	47	67
sky	91	91
window	66	71
wall	87	87
total accur.	78.0	**81.1**

3.7.5 Performance in facade parsing

To estimate the performance in facade parsing we evaluate our algorithm on the image dataset of Teboul et al. [59] against the ground truth proposed by Martinović et al. in [33] and compare our results to those of the three-layered parsing algorithm proposed by Martinović et al. [33]. The competing algorithm combines the results of a Recursive Neural Network (RNN), yielding classification of each pixel of the image, with window and door detections. We run our algorithm on the same RNN results as the authors of the competing method [33], both without detections and with window detections. Again, the detections are the same as the ones used to evaluate the three-layered method [33], but our method uses them in a different manner. We use the Haussmannian facade grammar presented in section 3.7.1. The grammar features more flexibility than the split grammar used in [59], but is less flexible than the set of 'weak architectural principles' used in [33]. For example, our grammar only models balconies that are as wide as the corresponding windows and ones that extend through the whole facade width. We also do not model doors. On the other hand, our algorithm guarantees the 'structural correctness' of the resulting segmentations, whereas the three-layered method [33] only imposes the 'weak architectural principles' locally, for example, by aligning nearby objects.

Even when run without the detection term, our graph grammar-based algorithm attains pixel-wise accuracy better than 'raw' RNN and close to the accuracy attained by the competing method. When detections are used, the performance grows and exceeds that of the three-layered method by a small margin. The reason for the superior performance of our algorithm is that it enforces a global alignment according to predefined patterns, encoded in a grammar. For example, if the bottom-up cues contain a single false positive balcony patch, that is not aligned with other balconies

Table 3.5: Performance on the ECP dataset with unary potentials obtained using a Recursive Neural Network (RNN). The rows corresponding to classes present class accuracy. The bottom row contains total pixel accuracy. In columns, starting from left: performance of the RNN; results of [33], based on these RNN results; results for our graph grammar-based algorithm with the RNN-based unaries without detection (GG*); our results for RNN-based unaries and detections (GG). We do not model doors.

	RNN	[33]	GG*	GG
roof	70	74	83	85
shop	79	93	92	92
balcony	74	70	72	72
sky	91	97	94	94
window	62	75	69	70
door	43	67	0	0
wall	92	88	90	91
pixel accur.	82.6	84.2	83.9	**84.8**

Figure 3.12: Comparison of parsing results obtained using our method and using the algorithm proposed in [33]. *Odd images*: our results, overlaid on the input images. Splitlines are in green, balconies are outlined in cyan and magenta. *Even images:* results reported in [33]. Note the artifacts: some windows are wider than the corresponding balconies and unaligned balconies.

in the same floor, we ignore the patch. The 'weak architectural principles' used by the three-layered method do not enforce the global satisfaction of such constraints. Besides, the constraint factors in our models contain a lot of information about the relative sizes of facade elements, that are well tuned to the data. This prevents huge errors, like missing whole floors and leaving large wall areas. Finally, even though we do not model doors, nor balconies that extend outside of windows, but do not extend through whole facade with, these elements actually happen to take a small fraction of the facade area, and are difficult to capture. That is, even when modeled, they are often segmented erroneously, as shown in table 3.5.

The strong benefit of our method is that it guarantees that the resulting segmentations belong to the language generated by the grammar and thus satisfy strong architectural constraints. Artifacts generated by the three-layered method, like the ones shown in figure 3.12, are thus avoided.

We also test the proposed method against another state-of-the-art facade segmentation scheme, proposed by Cohen et al. [9]. We obtain the per-pixel potentials

Table 3.6: Performance on the ECP dataset with unary potentials obtained using TextonBoost. The rows corresponding to classes present class accuracy. The bottom row indicates the total pixel accuracy. In columns, starting from left: performance of the 'raw' classifier, obtained by assigning the most likely class independently for each pixel; results of Cohen et al. [9]; results for our graph grammar-based algorithm with the texture classification-based potentials without detection (GG*); our results for potentials combining texture classification and detections (GG). We do not model doors.

	raw	[9]	GG*	GG
roof	89	90	80	85
shop	95	94	96	96
balcony	90	91	84	83
sky	94	97	96	96
window	86	85	80	77
door	77	79	0	0
wall	90	90	86	86
pixel accur.	90.1	**90.8**	85.3	85.5

using the method described in their paper. They apply a variant of TextonBoost, implemented by the authors of [29]. They use SIFT and Color SIFT descriptors, Local Binary Patterns and location features. The features of each type are clustered using K-means into 512 clusters. They establish a neighborhood of 200 random rectangles and the final feature vector is a concatenation of histograms of cluster members' appearance in these rectangles. The per-pixel costs c_{ijt} result from multi-class boosting [50]. Again we detect windows using the Viola-Jones detector, as in section 3.7.3. The results are presented in table 3.6.

Our algorithm performs worse than the competing method and the 'raw' segmentation obtained by assigning the most probable class independently to each pixel. One reason for that is that the texture classifier already gives very good performance and the constrained expressive power of our grammar, limiting the balcony types, starts to influence the results. Another factor is that the randomized search for optimal structure does not always explore and sample the exact optimal grammar derivation, missing the optimal structure. The explanation of the negligible gain from using detections is that they do not add much information to the excellent results of texture classification.

One advantage of our method over the algorithm of Cohen et al. though is that we enforce simultaneous alignment in two dimensions, which the competing algorithm does not handle.

3.8 Conclusion

We have presented a novel method for modeling complex objects, where model structure and part positions are optimized separately. The validity of this approach has been confirmed experimentally. We have also compared the performance of our method to that of two existing state-of-the-art algorithms, shown comparable and slightly lower performance. Another experiment shows that the performance is partly due to the proposed scheme of fusing texture classification and object detec-

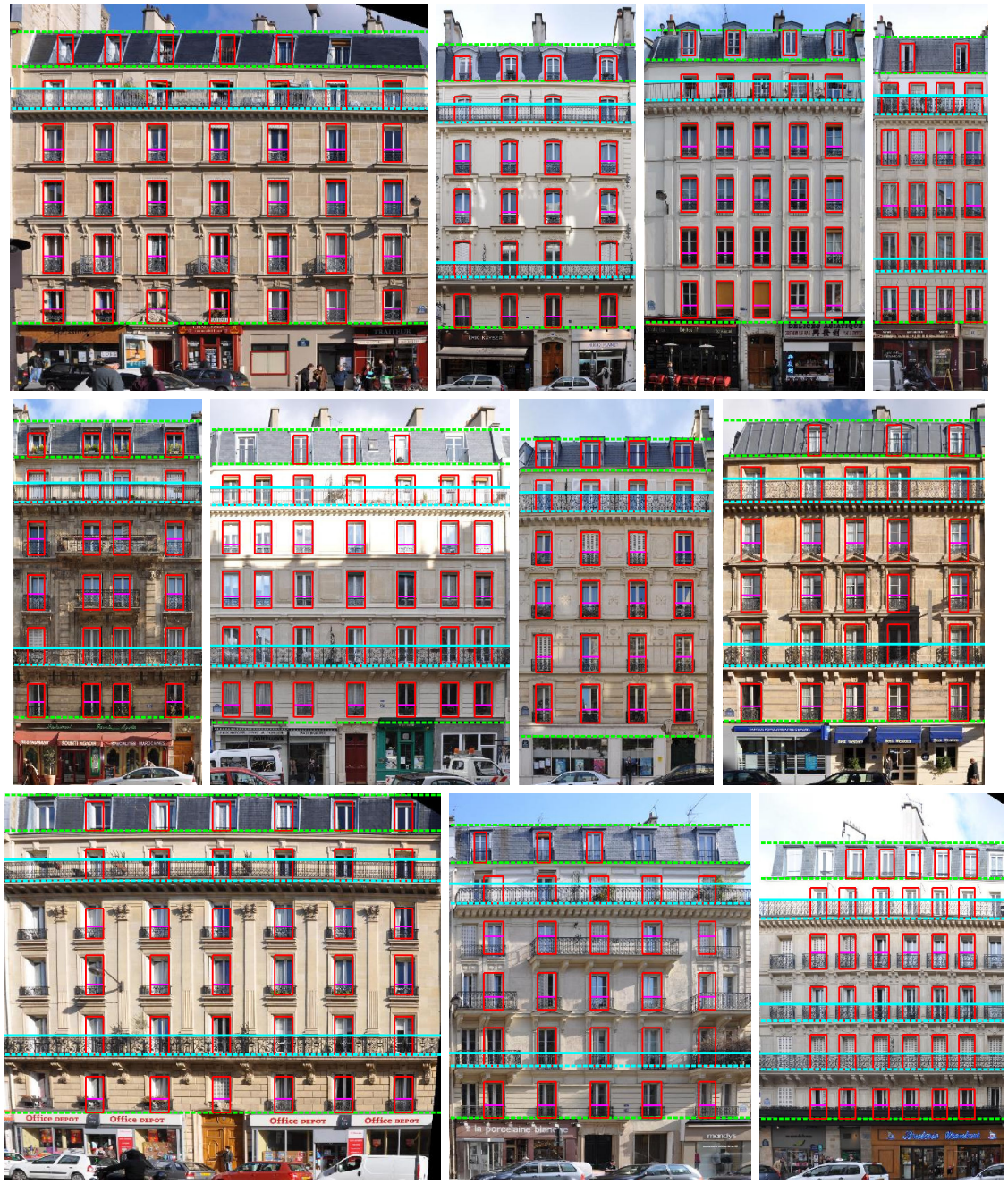


Figure 3.13: Example results of the proposed algorithm. Green lines separate sky from roof, roof from facade and facade from shop. Balconies are outlined in magenta or cyan. Note the variety of alignment patterns supported by the algorithm (top right). Typical errors are presented in the bottom row. In these images window columns can be missed due to the randomized, non-exhaustive search of the structure space. In the second and third images, our grammar does not model complex-enough patterns of the facade, like vertical misalignment of windows and middle-size balconies.

tions. An important advantage of our method over the grammar-free segmentation methods is that it only produces results that belong to the specified language, and thus does not produce 'structurally invalid' segmentations.

The proposed algorithm is a step towards more efficient facade parsing algorithms, that are not based on random exploration of the problem space. Even though in this case the random exploration was limited to the search for a model structure and not eliminated completely, this already leads to better segmentations than with algorithms based only on random exploration.

This work raises a number of challenges, including finding efficient bottom-up methods of building the initial structure graph, methods for warm-starting inference in a graphical model after perturbing its structure, modeling projections of 3D objects by means of graph grammars and learning grammars from training data.

It must however be noted that this graph-grammar based approach has some inherent limitations:

- it is difficult to come up with a principled method of estimating the structure of the facade, that is not based on sampling graphs from the grammar;
- specification of graph-grammar priors is tedious and unintuitive for a human user, and learning a graph grammar from image annotations represents a difficult problem in itself;
- approximating the optimal part positions for a given fixed structure is a time-consuming operation that needs to be repeated for each sampled structure.

Moreover, both the complexity of the grammar and the computational cost of approximating optimal part positions for a fixed structure grow rapidly with the modeled level of architectural details. For example, we do not model balconies that begin and end at an arbitrary point along the floor; we only model balconies constrained to window cavities and balconies running through the whole width of the facade. While modeling these medium size elements is technically possible, a graph grammar that represents them would be complex and the number of alternatives would be very large, drastically increasing the size of the combinatorial space of structures and the time of optimization for a fixed structure.

In the following chapters we propose a different approach that addresses some of the mentioned limitations. In particular, we allow more structural variation (irregular balconies) and we formulate the whole problem of parsing, including structure and positions of parts, as an optimization problem that does not require sampling.

Publications

The work presented in this chapter has been published in

Koziński, M., Marlet, R. (2014). Image parsing with graph grammars and Markov Random Fields applied to facade analysis. In *2014 IEEE Winter Conference on Applications of Computer Vision (WACV)*.

Chapter 4

Bidirectional Alignment by Labeling Image Rows and Columns

4.1 Introduction

As stated in chapter 2.1.2, finding the most likely facade segmentation, consistent with a user-defined grammar-based shape prior, requires a random exploration of a high dimensional space of grammar derivations. The random exploration-based algorithms cannot be relied on to repeatedly produce optimal results. In consequence, existing algorithms suffer from the 'curse of structural exploration'. In chapter 3 we presented a method to limit this random exploration to the space of facade structures, while formulating the problem of inferring positions of facade elements as MAP-MRF inference and solving it using a state-of-the-art solver. Although the proposed algorithm leads to better segmentations than the split grammar-based approach [60, 59, 47], it is still affected by the curse of structural exploration. The need to randomly explore the space of facade structures prevents us from formulating complex grammars, expressing rich architectural variation with a large number of alternatives.

In this chapter we propose a facade segmentation framework, in which the random exploration is eliminated completely. This is done by modifying the split grammar formalism in a way that lets us formulate parsing in terms of assigning classes to image pixels, rows and columns. The formulation takes the form of a binary linear program. Eliminating the random exploration lets us directly search for approximations of the globally optimal segmentations, both in terms of facade structure and accurate layout of its elements. We approximate the optimal segmentations using the dual decomposition (DD) algorithm frequently applied to optimization problems arising in computer vision [26, 54]. Our algorithm attains state-of-the-art performance on standard datasets.

4.1.1 Contributions

Our novel approach to image parsing lifts the curse of procedural exploration. It is based on a new formulation of shape priors that allows expressing the problem of optimal segmentation as a binary linear program.

Instead of expressing priors using grammar rules (e.g., $A_0 \rightarrow A_1 \dots A_n$), which tightly merge immediate dominance (decomposition of object A_0 into sub-objects A_i's) and immediate precedence (spatial left-adjacency of A_i to object A_{i+1}), we separate the two aspects: the user specifies on one hand an unordered tree hierarchy representing a structural decomposition of the scene (e.g., where A_0 is decomposed into a configuration of A_i's), and on the other hand a set of forbidden or unlikely spatial neighbor pairs (e.g., $A_{i+1} \not< A_i$, meaning A_{i+1} cannot be left-adjacent to A_i). We also relax object instantiation. Contrary to a right-hand side of a grammar rule, our prior does not specify the number of instances of objects of a given type (e.g., n symbols on the right hand side of the production presented in the beginning of this paragraph). Specific instance configurations can be forbidden though, by specifying the forbidden neighbor pairs. Last, we make the similarity of object instances a part of the formal framework. For example, two floors of the same class are expected to have exactly the same structure, with vertically aligned windows. This new formalism enables the specification of complex architectural structures, including interleaved grid alignments. For practical purposes, its expressive power is similar to that of grammar-based formalisms.

The problem of parsing with a prior of the proposed form can be turned into a linear binary program, which we solve efficiently using dual decomposition, thus eliminating the need for a procedural exploration of the solution space. As shown in the experiment section, our algorithm yields repeatable, state-of-the-art performance on standard datasets. Our model actually combines the accuracy of methods using hard-coded structural constraints [33, 9] with the flexibility of grammar-based methods [59, 47]. Abstract comparison to the state of the art is summarized in table 4.1.

Table 4.1: Comparison of the proposed algorithm with state-of-the-art facade parsing methods. The split grammar parsing framework by Teboul et al. [59] relies on an unreliable, sampling-based optimization scheme. The three-layered method proposed by Martinović et al. [33] applies only 'local' corrections to segmentation results according to hard-coded principles. The dynamic programming-based algorithm of Cohen et al. [9] is based on a predefined sequence of dynamic programs, that restrict possible facade structures, but do not enforce grid alignment. The 2D version of a CYK parsing algorithm proposed by Riemenschneider et al. [47] requires a severe image subsampling. Our graph grammar-based algorithm (GG), presented in chapter 3, requires inefficient sampling of facade structures from a grammar. The proposed formulation based on row and column labels (RCL) allows us to directly approximate optimal segmentations.

Property	[59]	[33]	[9]	[47]	GG	RCL
User-defined shape prior	✓	–	–	✓	✓	✓
Approximation of global optimum	–	–	–*	✓	–	✓
No need of image subsampling	✓	✓	✓	–	✓	✓
Simultaneous alignment in two dimensions	✓	✓	–	✓	✓	✓

* Cohen et al. [9] can issue a certificate of optimality if the found solution is optimal.

Table 4.2: Symbols used in this chapter

h	image height in pixels; defined on page 59
w	width of an image in pixels; defined on page 59
I	the set of image row indexes $I = \{1, \dots h\}$; p. 59
J	the set of image column indexes $J = \{1, \dots w\}$; p. 59
$\mathcal{I}$	the set of indexes of image pixels, $\mathcal{I} = I \times J$; p. 59
$\mathcal{C}$	the set of rectangle classes; p. 60
K	the subset of rectangle classes $K \subset \mathcal{C}$, called row-classes; p. 60
L	the subset of rectangle classes $L \subset \mathcal{C}$, called column-classes; p. 60
r	the rectangle class corresponding to the root of the tree of classes; p. 60
$Ch(n)$	the set of classes that are children of class n in the tree of classes; p. 60
$Pa(n)$	the parent class of class n in the tree of classes; p. 60
$Sib(n)$	the set of sibling of n in the tree of classes (the set of classes that have the same parent as n in the tree); p. 60
$Anc(n)$	the set of ancestors of class n in the tree of classes; p. 60
$Desc(n)$	the set of descendants of class n in the tree of classes; p. 60
T	the set of classes corresponding to leaves in the tree of classes; p. 60
x_{jl}	a variable encoding assignment of column class $l \in L$ to image column $j \in J$; p. 60
$\mathbf{x}_j$	a vector of $(x_{jl})_{l \in L}$ for a given column j; p. 60
$\mathbf{x}$	a vector resulting from stacking all x_{jl}, $\mathbf{x} = (x_{jl})_{j \in J, l \in L}$; p. 66
y_{ik}	a variable encoding assignment of row class $k \in K$ to image row $i \in I$; p. 60
$\mathbf{y}_i$	a vector of $(y_{ik})_{k \in K}$ for a given row i; p. 60
$\mathbf{y}$	a vector resulting from stacking all y_{ik}, $\mathbf{y} = (y_{ik})_{i \in I, k \in K}$; p. 66
L_k	the subset of column classes $L_k \subset L$, containing all children of row class $k \in K$ in the tree, and all column class siblings of ancestors of k in the tree; the characteristic feature of the set is that exactly one of its components has to be assigned to each image column; p. 64
K_l	the subset of row classes $K_l \subset K$, containing all children of column class $l \in L$ in the tree, and all row class siblings of ancestors of l in the tree; the characteristic feature of the set is that exactly one of its components has to be assigned to each image row; p. 64
$y^l_{ikk'}$	a variable encoding assignment of class k to row i and class k' to row $i+1$; for technical reasons the variable is defined for $l \in L \setminus T, k, k' \in K_l$; p. 65
$x^k_{jll'}$	a variable encoding assignment of class l to column j and class l' to column $j+1$; for technical reasons the variable is defined for $k \in K \setminus T, l, l' \in L_l$; p. 65
$c_{ll'}$	the cost of assigning classes l and l' to neighboring image columns; p. 66
$c_{kk'}$	the cost of assigning classes k and k' to neighboring image rows; p. 66
z_{ijt}	a variable encoding assignment of class $t \in T$ to pixel $(i, j) \in \mathcal{I}$; p. 65
$\mathbf{z}$	a vector created by stacking all variables z_{ijt}; p. 66
c_{ijt}	the cost of assigning class $t \in T$ to pixel $(i, j) \in \mathcal{I}$; p. 66

z^k_{ijt}, z^l_{ijt} copies of variables z_{ijt} used in the slave problems, defined for $k \in K \setminus T$ and $l \in L \setminus T$; p. 121

x^k_{jl} a copy of variable x_{jl} used in a slave indexed by k; the variable is defined for $k \in K \setminus T$ and $l \in L_k$; p. 121

y^l_{ik} a copy of variable y_{ik} used in a slave indexed by l; the variable is defined for $l \in L \setminus T$ and $k \in K_l$; p. 121

$\mathbf{z}^k$, $\mathbf{z}^l$ vectors of variables z^k_{ijt} and z^l_{ijt}, respectively, for $(i,j) \in \mathcal{I}$ and $t \in T$; p. 121

$\mathbf{x}^k$ a vector of variables x^k_{jl} for all $j \in J$ and $l \in L_k$; p. 121

$\mathbf{y}^l$ a vector of variables y^l_{ik} for all $i \in I$ and $k \in K_l$; p. 121

$\breve{\mathbf{z}}$ a vector of all z_{ijt}, z^k_{ijt} and z^l_{ijt}; p. 123

$\breve{\mathbf{x}}$ a vector of all x_{jl}, x^k_{jl} and $x^k_{jll'}$; p. 123

$\breve{\mathbf{y}}$ a vector of all y_{ik}, y^l_{ik} and $y^l_{ikk'}$; p. 123

λ^l_{ijt}, λ^k_{ijt} dual variables corresponding to constraints coupling variables z^k_{ijt} and z^l_{ijt}; p. 123

$\boldsymbol{\lambda}^l$, $\boldsymbol{\lambda}^k$ the vectors resulting respectively from stacking all λ^l_{ijt} and λ^k_{ijt}; p. 123

$\boldsymbol{\lambda}$ the vectors resulting from stacking all λ^l_{ijt} and all λ^k_{ijt}

γ^k_{jl} a dual variable coupling x^k_{jl} for different k; it is only defined for $l \in L_k$; p. 123

$\boldsymbol{\gamma}^k$ the vector resulting from stacking all γ^k_{jl} for $j \in J$ and $l \in L_k$; p. 123

γ^l_{ik} a dual variable coupling y^l_{ik} for different l; it is only defined for $k \in K_l$; p. 123

$\boldsymbol{\gamma}^l$ the vector resulting from stacking all γ^l_{ik} for $i \in I$ and $k \in K_l$; p. 123

$\boldsymbol{\gamma}$ the vector resulting from stacking all γ^k_{jl} and γ^l_{ik}

$(\hat{\mathbf{z}}^k, \hat{\mathbf{x}}^k)$ the optimal argument of a slave indexed by $k \in K \setminus T$; p. 123

$(\hat{\mathbf{z}}^l, \hat{\mathbf{y}}^l)$ the optimal argument of a slave indexed by $l \in L \setminus T$; p. 124

α^n stepsize in the n-th iteration of the algorithm; p. 125

$\bar{z}^n_{ijt}$ the mean value of optimal arguments $\hat{z}^{l,n}_{ijt}$ and $\hat{z}^{k,n}_{ijt}$ of the slaves in iteration n; p. 125

$\bar{y}^n_{ik}$ the mean value of optimal arguments $\hat{y}^{l,n}_{ik}$ of the slaves indexed with $l \in L \setminus T$; p. 125

$\bar{x}^n_{jl}$ the mean value of optimal arguments $\hat{x}^{k,n}_{jl}$ of the slaves indexed with $k \in K \setminus T$; p. 125

T^*_l the set of terminal classes that do not descend from any class $k \in K_l$; p. 127

δ_{ijk} an auxiliary variable; p. 127

t^k_{ij} the optimal class of pixel (i,j), given the class k of row i; p. 128

$\tilde{c}_{ijk}$ the optimal cost of assigning a class to pixel (i,j) by a slave, given the class k of row i; p. 128

t^*_{ij} the lowest-cost class from the set T^*_l of pixel (i,j); p. 128

$\tilde{c}^*_{ij}$ the lowest cost of assigning a class from the set T^*_l to pixel (i,j); p. 128

c^l_{ik} the cost of asigning class k to row i by slave l; p. 129

$\phi(i,k)$ the optimal cost of labeling i first rows by a slave, given that the i-th row gets label k; p. 129

$\kappa(i,k)$ the optimal class of row i given that row $i+1$ gets class k; p. 129

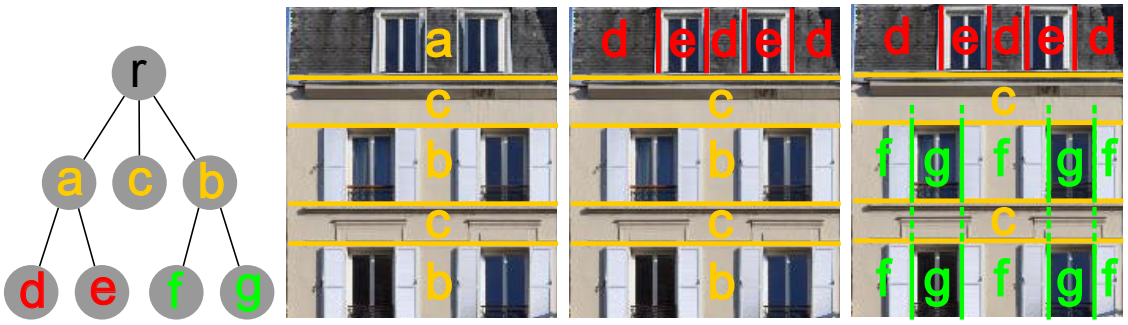


Figure 4.1: The shape prior consists of a hierarchy of rectangle classes (image 1 on the left) and a table of pairwise potentials for each nonterminal node (not shown here). Each image (2-4) shows the substitution of all rectangles of a particular class with any number of rectangles of any child classes. The class denoted by r, corresponding to the root of the hierarchy, represents the whole image. The remaining classes are: a-attic, b-floor, c-wall between floors, d-roof, e-attic window, f-wall between windows, g-window. The two-dimensional alignment of windows in the facade (but not attic windows) is enforced by the requirement that all floors are split in the same way.

4.2 Proposed model

Although it departs from grammar-based approaches, our structural segmentation framework is inspired by split grammars.

The shape prior of our split-based segmentation is encoded as a tree with nodes corresponding to classes of rectangular image regions, created by image splitting and called rectangles further in the text. Child nodes represent classes of rectangles resulting from splitting a rectangle of a parent class. We require that a rectangle of a class resulting from a vertical split can only be split horizontally and vice versa. Consequently, all non-leaf nodes at a given tree depth are split along the same direction. This tree is complemented with a table of pairwise potentials associated to each non-leaf node. The pairwise costs can be used to penalize invalid or unlikely adjacency configurations of child rectangles. Our algorithm can handle infinite values of the potentials and in our experiments we only use binary potentials that take the value of zero or infinity, fully preventing some configurations of neighbors and allowing the others.

In contrast to classical split grammars, which are context-free and cannot be used to express simultaneous alignment in two dimensions (other than with implementation tricks that introduce context dependency), we require that rectangles of the same class are aligned both vertically and horizontally. This requirement can be enforced by constraining all rectangles of the same class that are aligned along the splitting direction to be split in the same positions into subrectangles of matching classes. An example tree and corresponding segmentations are presented in figure 4.1. Note the bidirectional alignment of windows (class g).

4.2.1 Optimal Segmentation as a Binary Linear Program

We denote the set of indexes of image pixels by $\mathcal{I} = \{(i,j)|i \in I, j \in J\}$, $I = \{1,\ldots,h\}$ and $J = \{1,\ldots,w\}$, where h is image height and w is image width. That is, the first pixel index identifies the corresponding image row, and the second one

specifies the column [1]. We denote the set of rectangle classes by $\mathcal{C} = K \cup L$, where K denotes the set of classes that result from a horizontal split, also called row-classes, and L is the set of classes that result from a vertical split, called column-classes, and $K \cap L = \emptyset$. The root r of the tree of classes is a 'starting class', corresponding to the whole image. Without loss of generality we assume that r is split horizontally and by convention we consider $r \in L$. We recall that nodes in K can only have children in L and vice versa. Consequently, all nodes at each level of the tree are either column-classes or row-classes. We denote the set of children of class $n \in \mathcal{C}$ by $Ch(n)$ and the set of descendants of n, including n, by $Desc(n)$. Similarly, we denote the set of ancestors of n, including n, by $Anc(n)$, and its parent by $Pa(n)$. The set of siblings of n is denoted $Sib(n)$. We define the set of classes $t \in \mathcal{C}$ corresponding to the leaves of the tree by T and call its members terminal classes.

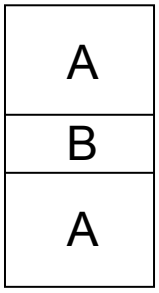
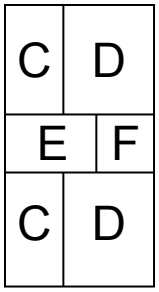
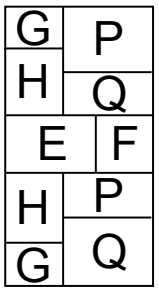
For all $i \in I$, $j \in J$, $k \in K$ and $l \in L$, we define variables $y_{ik}, y_{il}, x_{jk}, x_{jl} \in \{0, 1\}$ such that $y_{ik} = 1$ if k may be present in row i and $x_{jl} = 1$ if l can appear in column j. We make a distinction between the variables encoding assignment of row-classes $k \in K$ and column-classes $l \in L$, because they behave differently for horizontal and vertical splits. A sequence of vertical and horizontal splits assigns a sequence of rectangle classes to every pixel of the image. The sequence of classes forms a path from the root to a leaf in the tree of classes. For any image row i, it is then possible to list all the classes that are assigned to at least one pixel on the row. Below we show that a segmentation consistent with a prior of the proposed class tree form can be represented in terms of the sets of classes assigned to each image row and column, encoded by variables y_{ik}, y_{il}, x_{jk} and x_{jl}. This row- and column-based formulation enables global alignment of distant rectangles of the same class.

In table 4.3, we present how the sets of row- and column-classes that are present in image rows and columns are manipulated in the process of shape derivation. We also formulate constraints on y_{ik}, y_{il}, x_{jk} and x_{jl} that reflect this behavior. The first row of the table illustrates the first split, horizontal by convention. After the split, only the children of the root class r may appear in every column of the image. Equation (4.1) encodes that in terms of variables x_{jl}. Exactly one of the children appears in each image row. This is encoded by equation (4.2). As shown in the second row of the table, in case of a vertical split only one child rectangle is going to appear in each image column previously occupied by the parent. We emphasize that all rectangles of the same class are split simultaneously along the same lines, so that the child rectangles are aligned and their classes are consistent along the splitting axis. For example, in the table, the two rectangles of class A are split in the same positions into vertically aligned rectangles of classes C and D. Therefore, a vertical split introduces precisely one child class to each column in which the parent class was present. Equation (4.3) in table 4.3 encodes this statements formally in terms of variables x_{jl}. However, all child classes are going to appear in each image row where the parent was present. This is encoded by equation (4.4). Analogical reasoning applies to horizontal splits. The corresponding constraints on child and parent classes assigned to a row or column are encoded by equations (4.6) and (4.5).

From equation (4.4) in the fourth column of table 4.3 it follows that, for vertical

[1] In this chapter we use this row-column indexing, consistent with image indexing in Matlab.

Table 4.3: Illustration of the splitting process and interpretation of the variables x_{jl} and y_{ik}. The splitting process is just a concept that helps us to introduce our formulation and not a mode of operation of the proposed algorithm.

	Example sequence of splits	Specific constraints for the example	General constraints for any class tree	
First split $r \to \{A, B\}$		$x_{jA} = 1$ $x_{jB} = 1$ $y_{iA} + y_{iB} = 1$	$\forall k \in Ch(r),\ x_{jk} = 1$ $\sum_{k \in Ch(r)} y_{ik} = 1$	(4.1) (4.2)
Vertical splits $A \to \{C, D\}$ $B \to \{E, F\}$		$x_{jC} + x_{jD} = x_{jA}$ $x_{jE} + x_{jF} = x_{jB}$ $y_{iA} = y_{iC} = y_{iD}$ $y_{iB} = y_{iE} = y_{iF}$	$\forall k \in K,\ \sum_{l \in Ch(k)} x_{jl} = x_{jk}$ $\forall k \in K, \forall l \in Ch(k),\ y_{il} = y_{ik}$	(4.3) (4.4)
Horiz. splits $C \to \{G, H\}$ $D \to \{P, Q\}$		$x_{jC} = x_{jG} = x_{jH}$ $x_{jD} = x_{jP} = x_{jQ}$ $y_{iC} = y_{iG} + y_{iH}$ $y_{iD} = y_{iP} + y_{iQ}$	$\forall l \in L, \forall k \in Ch(l),\ x_{jk} = x_{jl}$ $\forall l \in L,\ \sum_{k \in Ch(l)} y_{ik} = y_{il}$	(4.5) (4.6)

splits, the state of each y_{il} for $l \in Ch(k)$ is determined by y_{ik}. According to equation (4.5) the same holds for horizontal splits, x_{jk} and x_{jl}. We eliminate x_{jk} and y_{il} by expanding equation (4.5) in (4.3) and equation (4.4) in (4.6). We get

$$\forall i \in I, \forall l \in L \setminus (T \cup \{r\}),\ \sum_{k' \in Ch(l)} y_{ik'} = y_{iPa(l)}, \tag{4.7a}$$

$$\forall j \in J, \forall k \in K \setminus T,\ \sum_{l' \in Ch(k)} x_{jl'} = x_{jPa(k)}. \tag{4.7b}$$

Each constraint of type 4.7a involves the parent of some class and the children of this class in the class tree. To aid understanding the structure of the constraint, we visualize its domain in fig. 4.2.

In the interest of maintaining the convention of assigning row-classes $k \in K$ to rows and column-classes $l \in L$ to columns, we modify constraints (4.1) and (4.2), from the first row of the table. We require that the root class is assigned to each

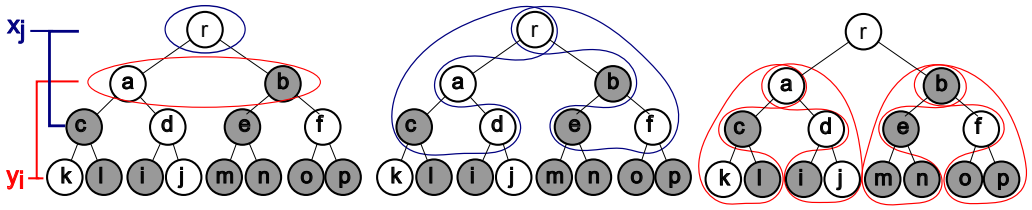


Figure 4.2: Visualization of the state of variables y_{ik} and x_{jl} for some pixel (i, j), and an example class tree (different from the ones in figures 4.1 and 4.3) The white nodes correspond to row- and column-classes k and l for which $y_{ik} = 1$ and $x_{jl} = 1$. The gray nodes correspond to classes k and l with $y_{ik} = 0$ or $x_{jl} = 0$. The domains of constraints of type (4.7) on y_{ik} are circled in blue and the domains of the constraints on x_{jl} are circled in red. Left: the domains of (4.7c). Middle and right: the domains of (4.7a). Note that only one leaf is connected to the root by a path of white nodes. This illustrates the uniqueness of pixel class given the state of variables corresponding to its row and column, stated in lemma 1.

column and that the first horizontal split assigns a unique class to each row:

$$\forall j \in J, \qquad x_{jr} = 1 \,, \tag{4.7c}$$

$$\forall i \in I, \quad \sum_{k \in Ch(r)} y_{ik} = 1 \,. \tag{4.7d}$$

Finally, in figure 4.3 we visualize the interpretation of vectors $\mathbf{x}_j = (x_{j'l})_{j'=j}$ for a fixed column j and $\mathbf{y}_i = (y_{i'k})_{i'=i}$ for a fixed row i, that satisfy constraints (4.7).

A key, albeit nontrivial consequence of constraints (4.7), is that each pixel is assigned exactly one class from each each level of the tree. The classes assigned to a pixel are unambiguously identified by the sets of classes present in the corresponding row and column.

Lemma 1. *Consider a hierarchy of classes given as a tree, as defined earlier. Denote the depth of the tree by M, the set of column-classes at the m-th level of the tree by L^m and the set of row-classes at the m-th level of the tree by K^m. Note that L^m is nonempty only for even m and K^m for odd m. Denote the vectors of $(y_{ik})_{i \in I, k \in K}$ and $(x_{jl})_{j \in J, l \in L}$ by $\mathbf{y}$ and $\mathbf{x}$. Denote the set of $\mathbf{y}$ and $\mathbf{x}$ satisfying constraint (4.7), enforcing the hierarchical structure of row-class and column-class assignment, by C_h. Then*

$$\begin{aligned}
(\mathbf{y}, \mathbf{x}) \in C_h \implies \quad & \forall (i,j) \in \mathcal{I} \quad \forall m \in \{0, \dots, M\}, \\
& \exists! l_{ij}^m \in L^m : \quad \forall n \in Anc(l_{ij}^m), \quad (x_{jn} = 1) \vee (y_{in} = 1) \quad \textit{if } m \textit{ is even} \qquad (4.8) \\
& \exists! k_{ij}^m \in K^m : \quad \forall n \in Anc(k_{ij}^m), \quad (x_{jn} = 1) \vee (y_{in} = 1) \quad \textit{if } m \textit{ is odd} \,. \qquad (4.9)
\end{aligned}$$

In words, for any $(i, j) \in \mathcal{I}$, for any values of y_{ik} and x_{jl}, that satisfy constraints (4.7), at any depth of the tree there exists exactly one row- or column-class such that the variables x_{jl} and y_{ik} corresponding to the class and all its ancestors are equal to one.

Proof. We prove the lemma by induction on the depth of the tree.

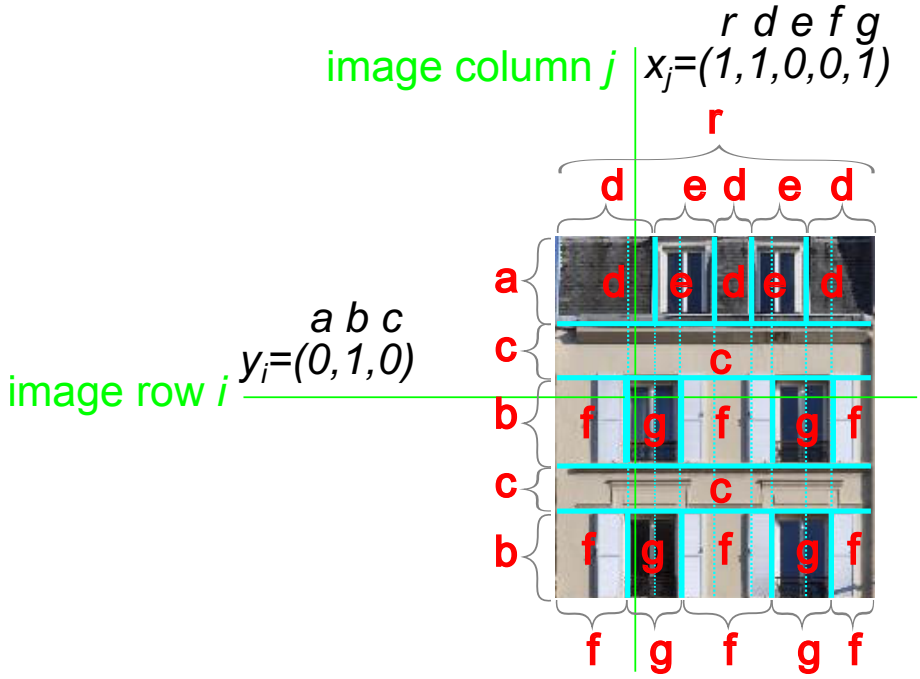


Figure 4.3: Interpretation of the vector $\mathbf{y}_i$ encoding row-classes assigned to row i, and the vector $\mathbf{x}_j$ of classes assigned to column j, satisfying constraints (4.7). Splitlines are drawn in cyan. The braces to the side of and above the image show the presence of particular classes (and their descendants) in image rows and columns. The image presents a segmentation corresponding to the full tree from figure 4.1.

The root r is the only node at depth $m = 0$ of the tree and, by constraint (4.7c), it holds that $x_{jr} = 1$ for all $j \in J$. Therefore the lemma holds for $m = 0$.

For depth $m = 1$ the tree is formed of the root and its children. By constraints (4.7d), we have that for each i there exists a single $k_i \in Ch(r)$ such that $y_{ik_i} = 1$ and $y_{ik} = 0$ for $k \neq k_i$. This proves the lemma for the case of a tree of depth 1.

Assume lemma 1 holds at depth m. If the level is of class $l \in L$, then by assumption for each i, j we have a single l_{ij}^m such that the variables associated to all its ancestors are equal one. By constraint (4.7a), exactly one child of l_{ij}^m will have its associated variable $y_{ik_{ij}^m}$ equal to one. Similar reasoning applies if the level is of row-class type. □

The constraints (4.7) allow assigning a number of row- and column-labels to each row and column. For a given column j, the number of possible configurations of $(x_{jl})_{l \in L}$ is combinatorial and grows exponentially with the number of vertically misaligned structures (rectangles independently split vertically, like attic a and floor b in figure 4.1) defined by shape prior. The same holds for feasible configurations of a set of row-classes assigned to image row. Below we show how the constraints can be reformulated into a more intuitive form. We transform the constraints by defining a number of non-disjoint subsets of L, such that exactly one class from each subset has to be assigned to each column. We do the same for the set of row classes K. First we present the definition of the subsets, and the new form of the constraints, and then we show the equivalence of the two formulations.

For each $k \in K \setminus T$ we define the set L_k as containing all children of k and all

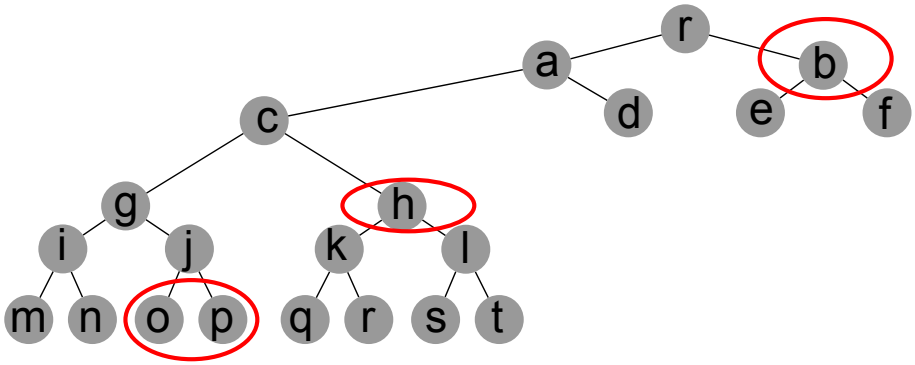


Figure 4.4: The structure of set K_l, for $l = \mathrm{j}$, visualized on a tree of classes. Members of the set are outlined in red. The nodes on depths 0, 2 and 4 of the tree represent classes $l \in L$ and nodes in depths 1, 3 and 5 correspond to classes $k \in K$. The characteristics of this set K_l is that exactly one of its members has to be assigned to each image row.

column-class siblings of ancestors of k. Formally,

$$L_k = Ch(k) \cup [L \cap (Ch(Anc(k)) \setminus Anc(k))] \,, \tag{4.10a}$$

where by $Ch(Anc(k))$ we denote the set of all children of all elements of $Anc(k)$. Similarly, for each $l \in L \setminus T$ we define

$$K_l = Ch(l) \cup [K \cap (Ch(Anc(l)) \setminus Anc(l))] \,. \tag{4.10b}$$

The structure of the sets is illustrated in figure 4.4. Using the sets K_l and L_k, constraints (4.7) can be represented as

$$\forall i \in I,\ \forall l \in L \setminus T \quad \sum_{k \in K_l} y_{ik} = 1 \,, \tag{4.11a}$$

$$\forall j \in J,\ \forall k \in K \setminus T \quad \sum_{l \in L_k} x_{jl} = 1 \,. \tag{4.11b}$$

Below we show how to transform constraints (4.7) to the new form (4.11) and vice versa.

The constraint (4.11a) indexed by $l = r$ has exactly the same form as (4.7d), since $K_r = Ch(r)$. For any $k \in Ch(r)$ the set $L_k = Ch(k)$ and the corresponding instance of constraint (4.11b) can be obtained by substituting (4.7c) into the instance of (4.7b) for the selected k. For classes more deep in the hierarchy obtaining a constraint of type (4.11) from constraints (4.7) requires a number of recursive substitutions. For a selected $k \in K \setminus T$ we substitute the left hand side of an instance of constraint (4.7b), indexed by k, into the left hand side of another constraint of the same form (4.7b), indexed by $Pa(Pa(k))$. We obtain an equation of the following form

$$\sum_{l \in Ch(k)} x_{jl} + \sum_{l \in Ch(Pa^2(k)) \setminus Pa(k)} x_{jl} = x_{jPa^3(k)} \,, \tag{4.12}$$

where $Pa^2(k) = Pa(Pa(k))$, etc. We repeat this process recursively, by plugging the left-hand side of (4.12) into the left-hand side of an instance of (4.7b), indexed by

$Pa^4(k)$, obtaining

$$\sum_{l \in Ch(k)} x_{jl} + \sum_{l \in Ch(Pa^2(k)) \setminus Pa(k)} x_{jl} + \sum_{l \in Ch(Pa^4(k)) \setminus Pa^3(k)} x_{jl} = x_{jPa^5(k)} \ . \tag{4.13}$$

We can recursively repeat the substitution until we get an equation with x_{jr} on the right-hand side. It is easy to see that, at this point, the sum on the left-hand side of the resulting equation contains all elements of the set L_k. Last, we plug the sum from the left-hand side of the developed equation into the left-hand side of constraint (4.7c), obtaining an equation of the form (4.11b). A similar series of recursive substitutions, ascending the hierarchy of classes, can be devised to get each of constraints (4.11a) from constraints (4.7).

To obtain from constraints (4.11) an equation of type (4.7b), indexed by k, we subtract an instance of (4.11b), indexed by $Pa^2(k)$, from an instance of (4.11b), indexed by k. Similar operation can be performed to obtain (4.7a) from (4.11a). As indicated before, the constraint (4.7d) has exactly the same structure as (4.11a) for $l = r$, since $K_r = Ch(r)$. Constraint (4.7c) cannot be recovered from constraints (4.11). However, the variables x_{jr} are only defined to conveniently terminate the hierarchy, and are of no practical importance. The feasible sets defined on variables $(x_{jl})_{l \in L \setminus \{r\}}$ and $(y_{ik})_{k \in K}$ by constraints (4.7) and (4.11) are the same.

We model the assignment of terminal classes to pixels by variables $z_{ijt} \in \{0, 1\}$, where $z_{ijt} = 1$ if pixel (i, j) is of class $t \in T$ and $z_{ijt} = 0$ otherwise. A single terminal class has to be assigned to each pixel

$$\forall (i, j) \in \mathcal{I}, \quad \sum_{t \in T} z_{ijt} = 1 \ . \tag{4.14}$$

By lemma 1, all ancestors of the class assigned to pixel (i, j) have the variables y_{ik} and x_{jl} equal to one, which leads to the following inequalities

$$\forall (i, j) \in \mathcal{I}, \forall k \in K, \sum_{t \in Desc(k)} z_{ijt} \leq y_{ik}, \tag{4.15a}$$

$$\forall (i, j) \in \mathcal{I}, \forall l \in L, \sum_{t \in Desc(l)} z_{ijt} \leq x_{jl}. \tag{4.15b}$$

Each nonterminal class has a table of pairwise potentials defined on its children. The potentials determine the likelihood of observing neighboring rectangles of the child classes. We implement the potentials using variables encoding assignment of pairs of labels to pairs of rows or columns, in the same way as in linear formulations of Markov Random Fields [68]. Since that formulation can be used only when exactly one class is assigned to each data point, we turn again to sets K_l and L_k. We define variables $y^l_{ikk'}$ for each $l \in L \setminus T$ and $k, k' \in K_l$, and $x^k_{jll'}$ for each $k \in K \setminus T$ and $l, l' \in L_k$. Precisely, $y^l_{ikk'} = 1$ if and only if row i is assigned row-class k and row $i+1$ is assigned class k'. The constraints imposing consistency of the pairwise variables

with the per-column and per-row variables take the form

$$\forall i \in I \setminus \{h\}, \forall l \in L \setminus T,\ \forall k \in K_l \quad \sum_{k' \in K_l} y^l_{ikk'} = y_{ik}\ , \tag{4.16a}$$

$$\forall i \in I \setminus \{h\}, \forall l \in L \setminus T,\ \forall k' \in K_l \quad \sum_{k \in K_l} y^l_{ikk'} = y_{i+1\,k'}\ , \tag{4.16b}$$

$$\forall j \in J \setminus \{w\}, \forall k \in K \setminus T,\ \forall l \in L_k \quad \sum_{l' \in L_k} x^k_{jll'} = x_{jl}\ , \tag{4.16c}$$

$$\forall j \in J \setminus \{w\}, \forall k \in K \setminus T, \forall l' \in L_k \quad \sum_{l \in L_k} x^k_{jll'} = x_{j+1\,l'}\ . \tag{4.16d}$$

We denote the cost of assigning class t to pixel (i, j) by c_{ijt}, and the pairwise cost for column- and row-classes by $c_{kk'}$ and $c_{ll'}$. The pairwise costs are defined for pairs of classes that are siblings in the class tree. Note that for a pair of sibling classes k, k' there could possibly be more than one $l \in L \setminus T$ such that $k, k' \in K_l$. In consequence, we can have more than one variable modeling the assignment of the pair k, k' to neighboring rows $y^l_{kk'}$ indexed by different l. By convention, to penalize the transition between each pair of classes exactly once, we apply the pairwise cost $c_{kk'}$ to $y^l_{kk'}$ such that $k, k' \in Ch(l)$. We denote the vector resulting from stacking all z_{ijt} by $\mathbf{z}$, the vector of all x_{jl} by $\mathbf{x}$ and the vector of all y_{ik} by $\mathbf{y}$. The segmentation task can be formulated as minimizing the following objective

$$E(\mathbf{z}, \mathbf{y}, \mathbf{x}) = \sum_{(i,j) \in \mathcal{I}} \sum_{t \in T} z_{ijt} c_{ijt} + \sum_{i=1}^{h-1} \sum_{\substack{l \in L \setminus T \\ k,k' \in Ch(l)}} y^l_{ikk'} c_{kk'} + \sum_{j=1}^{w-1} \sum_{\substack{k \in K \setminus T \\ l,l' \in Ch(k)}} x^k_{jll'} c_{ll'} \tag{4.17}$$

subject to constraints (4.11) and (4.14) to (4.16). In the above formula we abuse the notation by dropping the pairwise variables $y^l_{ikk'}$ and $x^k_{jll'}$ from the list of arguments of the energy. The values of these variables are unambiguously determined by $\mathbf{x}$ and $\mathbf{y}$ through constraints (4.16).

4.3 Inference

The formulated problem is linear and has a large number of binary variables. The scale of our problem, that has one variable per pixel per class (not counting the per-row and per-column variables), and a similar number of constraints (4.15), makes the use of a general mixed integer programming (MIP) solver impractical due to excessively long running time. We therefore consider techniques that enable approximating the optimal solution in a reasonable time by exploiting the structure of the problem. We turn our attention to two Lagrangian relaxation techniques, dual decomposition (DD) [26, 54] and alternating direction of multipliers method (ADMM) [5]. We briefly characterize the methods and their main differences here to motivate our choice of algorithm, and introduce the selected technique in more details in appendix B. In both algorithms, we first relax the constraints on binary domains of the variables and let them vary within the range of $[0, 1]$. Then we relax

selected constraints, that is, we formulate a Lagrangian (DD), or augmented Lagrangian (ADMM), of the original problem with respect to the selected constraints. This is equivalent to removing the constraint and introducing to the objective function a new component that penalizes violation of the constraint. We then solve the dual problem induced by the (augmented) Lagrangian. The algorithm iteratively updates the dual solution in the direction of a subgradient. The constraint to be relaxed is selected in such a way, that the subgradient of the dual problem can be found by iteratively optimizing 'simpler' subproblems, that can be solved efficiently. We often say that the original problem 'is decomposed' into the subproblems. It can be shown that the optimum of the dual problem is a lower bound on the optimum of the original problem. Finally, from the dual solution we extract a primal discrete solution that approximates the optimum of the original problem.

The difference between DD and ADMM stems from the fact that the first formulation is based on Lagrangian and the latter one on augmented Lagrangian, that has an additional quadratic term. As a result ADMM can be shown to converge faster [5]. However, the subproblems in ADMM are quadratic. When using DD, our problem decomposes into linear subproblems that can be efficiently solved by means of dynamic programming. This motivates our choice of DD to solve the formulated problem. The experiments confirm that DD behaves well in our application.

We decompose the problem of optimizing (4.17), subject to (4.11) and (4.14–4.16), into one subproblem for each $l \in L \setminus T$ and one for each $k \in K \setminus T$, obtaining the number of subproblems equal to the number of non-leaf nodes of the tree of classes. Due to constraint (4.11), a solution to each subproblem assigns exactly one row-class to each image row or exactly one column-class to each image column. The structure of all the subproblems is the same. First, optimal terminal classes of all pixels in all rows (columns) are determined, for all possible classes of each row (column), according to constraint (4.15). Next, dynamic programming is used to determine the optimal row (column) class assignments. In the interest of readability we postpone the detailed presentation of the algorithm to appendix C.

4.4 Experiments

We tested the performance of our algorithm in facade parsing on two datasets. For each of them, we have created a shape prior consisting of a hierarchy of classes and, for each nonterminal node, a table of pairwise potentials encoding possible adjacency patterns of its children. As an example, the class hierarchy used in experiments on the Graz50 dataset (described below) is presented in figure 4.5. We use very simple, binary potentials, which penalize invalid ordering of rectangle classes, like sky under wall, with infinite cost. For each image, we run the DD algorithm for 100 iterations, with a fixed step-size sequence $\alpha_n = a/\sqrt{n}$, where n is the number of iterations and a is a constant. We implemented the algorithm in C++. The running time was about 4 minutes per single image of the ECP dataset. We run the experiments on a 6 core Core-i7 processor with 3GHz clock.

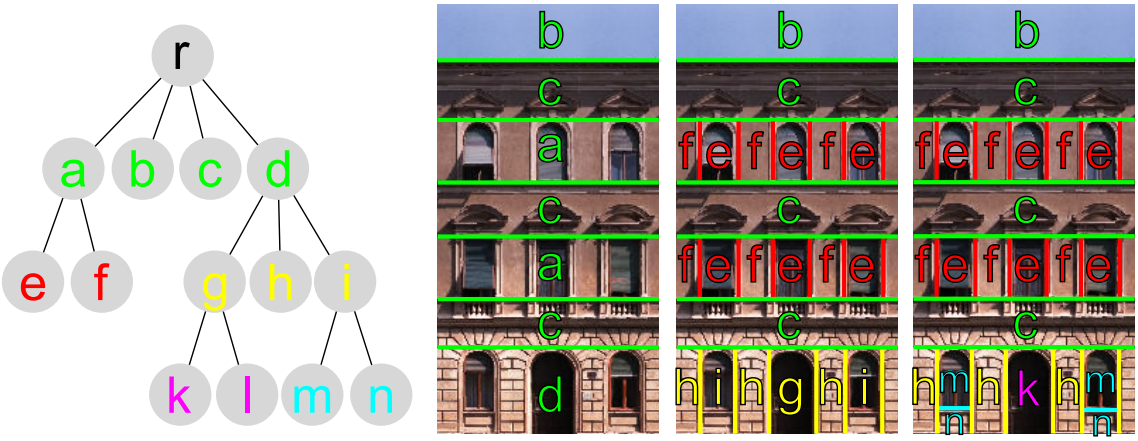


Figure 4.5: A prior used in experiments on the Graz50 dataset. The hierarchy of classes is presented on the left. The images illustrate interpretation of the classes. They contain segmentations corresponding to consecutive levels of the hierarchy. Class 'l', not used in the presented image segmentation, models a tile of wall above the door, and is used when windows in the ground floor are higher than top door boundary. We use pairwise potentials (not shown in the image) to forbid configurations where there is anything above the sky (class 'b') or below the door (class 'k').

4.4.1 Performance evaluation on the ECP dataset

We compare the performance of our algorithm to the method of Martinović et al. [33] on the ECP dataset [59], described in section 3.7. We use per-pixel costs generated by a Recursive Neural Network, identical to the one used in this baseline. We also compare the results against the outcome of graph grammar (GG) parsing on the same data. For all the comparisons we use the groundruth produced by Martinović et al. [33]. We follow the protocol used by Martinović et al. [33], using the results of RNN made available by the authors. The experiments were performed on five data folds with 60 training, 20 validation and 20 testing images (precisely, since the dataset contains 104 images, the first fold consists of 24 testing images, and the remaining four of 20 testing images). The RNN was trained on the 60 (resp. 64) training images of each fold and the validation set was not used in our experiment. The results are presented in table 4.4. The proposed method achieves better overall accuracy than the competing methods, even though both the baseline and our graph grammar-based approach use window detections in addition to RNN output. This lack of window detections results in the worse accuracy for that particular class. However, on average the linear programming-based method outperforms the other algorithms. We attribute the superior performance with respect to the method of Martinović et al. [33] to the use of a prior modeling 'global' alignment of distant objects, together with the proposed optimization scheme. Better performance in comparison to the graph grammar method follows from a richer prior, capable of modeling doors and balconies that start and end in arbitrary positions along the facade width (note the slightly better result for balconies in the table). Additionally, GG suffers from the necessity to sample facade structure, and if the correct structure is never drawn, the result is suboptimal. For example, see the failure cases with the incorrect number of window columns in figure 3.13.

We also compare the performance of our method to the dynamic programming-

Table 4.4: Performance on the ECP dataset [59] using RNN-based potentials as defined in [33]. The rows corresponding to classes present class accuracy (the diagonal entries of confusion matrices, or recall). The bottom row contains total pixel accuracies, that is, the percentage of correctly classified pixels. Starting from the left, we present the performance of Recursive Neural Network, then results of the method by Martinović et al. [33] which combines both the output of RNN and window, and door detections to obtain the per-pixel scores. In the third column, we present results of our graph grammar (GG) parsing with use of RNN scores and window detections (see chapter 3), and the last column contains the results the row and column labeling method (RCL) proposed in this chapter, using RNN-based per-pixel scores.

	RNN	[33]	GG	RCL
roof	70	74	85	83
shop	79	93	92	90
balcony	74	70	72	73
sky	91	97	94	92
window	62	75	70	57
door	43	67	0	40
wall	92	88	91	95
pixel accur.	82.6	84.2	84.8	**85.5**

based approach, proposed by Cohen et al. [9], which achieves state-of-the-art accuracy. We perform the comparison on the ECP dataset and against the ground truth produced by Martinović et al. [33]. We divide the dataset into five folds, the first of which consists of 24 test and 80 training images, and the remaining folds contain 20 test and 84 training images, respectively. The per-pixel energies are obtained according to the description presented in this baseline paper. They are generated by a multi-feature extension of TextonBoost, already described in section 3.7. The results are presented in table 4.5. Our results are similar to the ones presented by Cohen et al. [9]. This result is expected, as the shape priors used in the two cases have very similar expressive power and the authors of the baseline algorithm report that they obtained optimality certificates for 80% of the segmentations. Since our results are also close to optimality, the pixel-wise performance should be in both cases attributed to the performance of the texture classifier. The main advantage of our algorithm is that it can accept any user-defined shape prior on input, which makes it more general with respect to the competing method [9], which hard-codes the architectural constraints. Besides, our shape prior can express the alignment of architectural elements in two dimensions, which is beyond the expressive power of the dynamic programs [9].

4.4.2 Performance evaluation on the Graz50 dataset

The Graz50 dataset [47] is composed of 50 rectified images of building facades of different architectural styles. The facades feature more structural variation than the ones of the ECP dataset. The set of labels includes four classes: sky, wall, window and door. Performance on the Graz50 dataset has been tested using the TextonBoost pixel costs, obtained in the same manner as for the ECP dataset. Five folds were used and each time the dataset was split into 40 training images and 10

Table 4.5: Performance on the ECP dataset [59] with per-pixel potentials generated by TextonBoost. The rows corresponding to classes present class accuracy (the diagonal entries of confusion matrices, or recall). The bottom rows contain total pixel accuracy. Starting from the left, we present the results of assigning to each pixel the most probable class according to TextonBoost output (TB), results of the dynamic programming-based approach by Cohen et al. [9] using TextobBoost-based energies, then results of our graph grammar parsing (GG) ,and results of our row and column labeling-based method (RCL) using the same per-pixel costs.

	TB	[9]	GG	RCL
roof	89	90	85	91
shop	95	94	96	95
balcony	90	91	83	90
sky	94	97	96	96
window	86	85	77	85
door	77	79	0	74
wall	90	90	86	91
pixel accur.	90.1	**90.8**	85.5	**90.8**

Table 4.6: Performance on the Graz50 dataset [47]. The second and third columns of the table show class-wise accuracies resulting from the parsing algorithm proposed by Riemenschneider et al. [47], and the results of our method assigning row and column labels (RCL).

	[47]	RCL
sky	91	93
window	60	82
door	41	50
wall	84	96
total pixel accur.	78.0	**91.8**

test images. The results are presented in table 5.4. One reason why our results are superior to those in [47] is that their method requires a severe subsampling of the image to be tractable. Our method is more computationally efficient and can be run on full-resolution images.

4.5 Conclusion

We have presented a novel approach to grammar-based facade analysis in which the task of parsing is formulated as an integer program. Our formulation does not suffer from the curse of procedural exploration, that is typical for existing split grammar parsers. The proposed inference scheme enables practical application of more complex priors than the ones used before in experiments with split grammar parsing [60, 59] and in our graph grammar parsing method. For example, we were able to get highly accurate segmentations with a grammar modeling balconies that could be wider than a single window, but not run through the whole width of a facade. Our method attains the state-of-the-art performance in parsing on the ECP

Figure 4.6: Example parsing results on the ECP dataset (top and middle) and on the Graz50 dataset (bottom). Green lines separate sky from roof and roof from facade. Balconies are outlined in magenta, shops in yellow, and doors in cyan. Note the variety of alignment patterns supported by the algorithm (top right). Typical errors are missed doors and missed roof windows.

facade dataset and establishes a new state-of-the-art performance on the Graz50 dataset.

The complexity of the proposed formulation can be seen as a drawback. Learning how to write prior specifications can require some time. This can be problematic for a practical deployment of the method. The same can be said about studying the details of the formulation, which is disadvantageous from the point of view of further development of the proposed framework. Another disadvantage of the formulation is that priors created according to the specified format have low flexibility. This is a consequence of the global alignment principle stating that all regions of the same class should be aligned across the image. Modeling a facade element that is misaligned with other elements of the same type requires introducing a new class. For example, balconies on different floors of a Haussmannian building can be vertically misaligned. This forces us to have five different balcony classes, one for each floor of the facade. Finally, the running time of around 4 minutes per image also limits practical applicability of the algorithm.

In the next chapter, we propose an alternative formulation that is free of the above disadvantages, and keeps most of the advantages of the linear formulation.

Publications

The work presented in this chapter has been published in

Koziński, M., Obozinski, G., Marlet, R. (2014). Beyond Procedural Facade Parsing: Bidirectional Alignment via Linear Programing. In *Proc. 2014 Asian Conference on Computer Vision (ACCV)*.

Chapter 5

A Markov Random Field formulation of Facade Parsing with Occlusions

5.1 Introduction

In the previous chapter, we have proposed a framework that lifts the curse of structural exploration from facade parsing. However, the proposed framework suffered from a number of disadvantages resulting from a complex mathematical formulation. First, specification of priors required understanding the concept of the tree of classes, reflecting the splitting hierarchy, where vertical and horizontal splits alternate. The requirement of global two-dimensional alignment made it difficult to specify priors encoding patterns consisting of an unknown number of misaligned elements (for example, floors). The complex formulation made the method difficult to implement and resulted in long running times during inference. Additionally, existing frameworks that provide a systematic way for the user to define a shape prior, and yield segmentations consistent with the prior, are restricted to rectangular tilings. Furthermore, they are oblivious of occlusions. Even if they recover the occluded structure correctly, thanks to the prior information encoded in a grammar, they do not provide a way to segment the occluding object. The latter functionality is helpful, for example, in producing texture for building models produced from parsing results. For example, when the occluded regions are known, the texture in these regions can be inpainted using patterns from the non-occluded regions.

In this chapter, we propose a formulation which addresses these limitations. The key idea behind the new concept is encoding the shape prior in terms of pixel classes and constraints on classes assigned to neighboring pixels. This encoding enables formulating inference as finding the most likely configuration of a Markov Random Field (MRF) defined over a grid of pixels. Our formulation guarantees conformance of the resulting segmentations to the shape prior. The prior itself is expressed in a conceptually simple format. Priors expressed in terms of this new formalism couple elements of the same class less tightly than the ones presented in chapter 4, allowing more flexibility. Still, they enable modeling simultaneous alignment in two dimensions. Additionally, we propose a method of modeling elements of irregular shape and a way of handling occlusions. We model both the boundaries of the occluding object and the structure of the occluded facade. In a number of experiments, we

Table 5.1: Comparison of key properties of state-of-the-art facade parsing algorithms. GG: graph grammar algorithm presented in chapter 3, RCL: the algorithm for row and column labeling presented in chapter 4, AP: the adjacency pattern method presented in this chapter.

	[59]	[47]	[33]	[9]	GG	RCL	AP
User-defined shape prior	✓	✓	–	–	✓	✓	✓
Simultaneous alignment in 2D	✓	✓	✓	–	✓	✓	✓
No need of image subsampling	✓	–	✓	✓	✓	✓	✓
No need of sampling from a grammar	–	✓	✓	✓	✓	✓	✓
Occlusions and irregular shapes	–	–	✓	✓	–	–	✓

compare the performance of the proposed method to that of other algorithms on the same per-pixel potentials, showing better segmentations. The method achieves state-of-the-art results on several facade parsing datasets. The key features of the proposed framework, compared to the existing ones are summarized in table 5.1.

5.2 Related work

In this section we briefly review the literature related to the algorithm proposed in this chapter. Chapter 2 contains a more comprehensive review of existing work on facade parsing.

The shape prior proposed in this chapter is based on the concept of allowed and forbidden pairs of vertically and horizontally adjacent pixel classes. Interestingly, the same principle turned out to be useful for general image segmentation, as shown by Roy and Todorovic [48]. The input image is first oversegmented into superpixels. Then, a conditional random field is constructed over the superpixels. Additional mutex constraints, defined by the user, can be used to prohibit cooccurrence of a pair of labels in the same image, or occurrence of a certain pair of labels in any spatial configuration. The spatial configurations used in the experiments include above-below and left-right. While in the existing algorithm the mutex constraints can be formulated directly for classes of distant superpixels, in the method proposed in this chapter long-distance constraints are propagated by means of an auxiliary set of classes and constraints on classes of immediately adjacent pixels. In the existing work, a beam search technique, based on random exploration of the space of possible labelings, is used for inference. In the proposed framework, inference has the form of the MAP-MRF problem with hard constraints on classes of neighboring pixels. Finally, in our method the constraints are used to encode vertical and horizontal alignment, as opposed to enforcing constraints on classes of a pair of superpixels one of which is roughly to the right or left (above or below) from the other.

We propose to encode the irregular shape of segments, with monotonically increasing or decreasing boundaries, in terms of classes of vertically and horizontally adjacent pixels. This technique has been exploited by Felzenszwalb and Veksler [12] for segmenting images that feature a tiered structure, that is, are composed of three horizontal layers, with irregular boundaries. The middle layer can be subdivided vertically into a number of object classes. The authors propose an efficient dynamic programming algorithm for inference. The demonstrated applications include re-

covering layout of natural scenes and foreground-background segmentation. The constraint of the tiered scene structure makes the algorithm unsuitable for facade parsing.

Bai et al. [2] propose to use the constraints on classes assigned to vertically and horizontally adjacent pixels to model layout of indoor scenes. The model of a scene consists of five labels: corresponding to the floor, the ceiling, left and right walls, and the wall opposite to the camera. The model is based on the observation that, for example, the left wall has to be above the floor and below the ceiling. The authors formulate an efficient dynamic program for fitting the model to the image. The five-parts model is too limited to be directly useful for facade parsing.

Te problem of fitting the five parts model to an image can also be solved using the order-preserving moves [31]. The algorithm is a move-making method, based on two different moves, one shifting the labels vertically and the other, which shifts the labels along the horizontal direction, preserving the predefined order constraints. The authors show that the set of possible moves is larger than in the case of α-expansion and α-β swap, and therefore the order-preserving moves avoid some local optima of the latter methods. However, when the labels are supposed to form a 2D grid, like in the shape prior proposed in this chapter, alternating between optimization in the two directions is still prone to getting stuck in local optima.

Table 5.2: Symbols used in this chapter

h	image height in pixels; defined on page 87
w	width of an image in pixels; p. 87
I	the set of image row indexes $I = \{1, \dots h\}$; p. 85
J	the set of image column indexes $J = \{1, \dots w\}$; p. 85
$\mathcal{I}$	the set of indexes of image pixels, $\mathcal{I} = I \times J$; p. 85
$\mathcal{I}_h$	the set of indexes of all image pixels, except for the last column, $\mathcal{I}_h = I \times (J \setminus \{w\})$; p. 88
$\mathcal{I}_v$	the set of indexes of all image pixels, except for the last row, $\mathcal{I}_v = (I \setminus \{h\}) \times J$; p. 88
$\mathcal{G}$	a grid pattern; p. 78
$\mathcal{C}$	the set of column classes in a grid pattern specification; p. 78
$\mathcal{R}$	the set of row classes in a grid pattern specification; p. 78
$\mathcal{H}$	the set of pairs of column classes that can be neighbors in an image, a part of a grid pattern specification; p. 78
$\mathcal{V}$	the set of pairs of row classes that can be neighbors in an image, a part of a grid pattern specification; p. 78
A	an adjacency pattern; p. 78
S	the set of pixel classes in an adjacency pattern specification; p. 78
V	the set of ordered pairs of pixel classes that can be assigned to vertically adjacent pixels; a part of an adjacency pattern specification; p. 78
H	the set of ordered pairs of pixel classes that can be assigned to horizontally adjacent pixels; a part of an adjacency pattern specification; p. 78
$\hat{A}$	a hierarchical adjacency pattern; p. 84
$\mathcal{N}$	a set of nonterminal pixel classes in a hierarchical adjacency pattern; p. 84
$\mathcal{T}$	a set of terminal pixel classes in a hierarchical adjacency pattern; p. 84
N_0	the starting nonterminal class in a hierarchical adjacency pattern; p. 84
$\mathcal{P}$	the set of productions in a hierarchical adjacency pattern; p. 84
$Desc(s)$	the set of pixel classes descending from class s in a hierarchical adjacency pattern; p. 84
$Desc(p)$	the set of pixel classes descending from production p in a hierarchical adjacency pattern; p. 84
$Anc_p(s)$	the set single ancestor of class s in the hierarchy of adjacency patterns, in the pattern belonging to production p; it is only defined for $s \in Desc(p)$; p. 85
O	the set of classes of objects that can occlude a facade; p. 86
K	the set of semantic classes of facade elements (wall, window, etc.); p. 78
Ψ	a mapping assigning each pre-semantic class $s \in S$ a semantic class $k \in K$; p. 78

$\mathfrak{S}$	the set of pairs of pre-semantic and occluder classes, such that an object of the occluder class can occlude a region of the facade assigned the pre-semantic class; p. 86
$\phi_{ij\kappa}$	a cost of assigning a class $\kappa \in O \cup K$ to pixel (i, j); p. 88
$\theta(\sigma, \sigma')$	the pairwise cost of assigning classes σ and σ' to a pair of neighboring pixels; p. 82
$z_{ij\sigma}$	a variable encoding assignment of class σ to pixel (i, j); p. 88
$u_{ij\sigma\sigma'}$	a variable encoding assignment of class σ to pixel (i, j) and class σ' to pixel $(i, j + 1)$; p. 88
$\mathbf{u}$	the vector of all $u_{ij\sigma\sigma'}$; p. 88
$v_{ij\sigma\sigma'}$	a variable encoding assignment of class σ to pixel (i, j) and class σ' to pixel $(i + 1, j)$; p. 88
$\mathbf{v}$	the vector of all $v_{ij\sigma\sigma'}$; p. 88
$z'_{ij\sigma}$, $z''_{ij\sigma}$	copies of the variable $z_{ij\sigma}$ used by the slaves; p. 131
$\mathbf{z}'$, $\mathbf{z}''$	the vectors of all $z'_{ij\sigma}$ and $z''_{ij\sigma}$, respectively; p. 131
$\hat{\mathbf{z}}'$, $\hat{\mathbf{z}}''$	the optimal slave arguments
C', C''	the feasible sets of the slave problems; p. 132
$\lambda_{ij\sigma}$	a dual variable, corresponding to the constraint coupling $z'_{ij\sigma}$ and $z''_{ij\sigma}$; p. 132
$\boldsymbol{\lambda}$	a vector of $\lambda_{ij\sigma}$ for all $(i, j) \in \mathcal{I}$ and all $\sigma \in S_o$; p. 132
$\mathring{z}_{ij\sigma}$	a variable encoding the frequency of assignment of class σ to pixel (i, j); p. 133
$\mathring{\mathbf{z}}$	the vector of all $\mathring{z}_{ij\sigma}$; p. 133

5.3 Adjacency patterns as shape priors

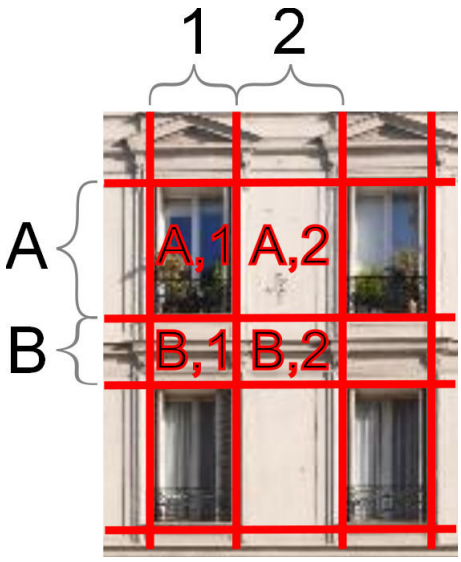
Simultaneous vertical and horizontal alignments are prevalent in facade layouts. To encode shape priors expressing such alignments, as well as more complex shapes, we introduce the notion of adjacency patterns.

5.3.1 From grid patterns to pixel adjacencies

Consider first the following simpler case of a shape prior encoding a grid pattern, which can be specified in terms of column and row classes. We denote the set of column classes by $\mathcal{C}$ and the set of row classes by $\mathcal{R}$. By assigning a column class $c \in \mathcal{C}$ to each image column, and a row class $r \in \mathcal{R}$ to each image row, we implicitly label each pixel with a pair (r, c) of a row class and a column class. We call such pairs of row and column classes $(r, c) \in \mathcal{R} \times \mathcal{C}$ the 'pre-semantic' classes. We define a set of 'semantic' classes K encoding types of facade elements (like wall, window, etc.), and a mapping Ψ that assigns to each pre-semantic class $(r, c) \in \mathcal{R} \times \mathcal{C}$ a semantic class $k \in K$. For facade parsing it is reasonable to prohibit some combinations of neighboring row or column classes. For example, segmentations where 'roof' is above 'sky' can be viewed as invalid. To encode such preferences, we can specify the set of ordered pairs of column classes that can be assigned to adjacent image columns $\mathcal{H} \subset \mathcal{C} \times \mathcal{C}$, and similarly the set of ordered pairs of adjacent row classes, $\mathcal{V} \subset \mathcal{R} \times \mathcal{R}$. We call a shape prior specification of the form $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{H}, \mathcal{V})$ a grid pattern. A simple example of a grid pattern, and a conforming segmentation, is presented in Figure 5.1.

We now introduce an alternative encoding of shape priors. We are going to show later that it is capable of expressing grid patterns as well as many more general, interesting priors. We define an 'adjacency pattern' as a triple $A = (S, V, H)$ where S is a finite set of (pre-semantic) classes, and $V \subset S \times S$ and $H \subset S \times S$ are sets of ordered pairs of classes that can be assigned to vertically and horizontally adjacent pixels. The condition $(s_1, s_2) \in V$ expresses that a pixel of class s_1 can be immediately below a pixel of class s_2. Similarly, the condition $(s_1, s_2) \in H$ allows a pixel of class s_1 to be immediately to the left from a pixel of class s_2. An ordered pair of labels $(s_1, s_2) \notin H$ cannot be assigned to a pair of horizontally adjacent pixels in such a way that the left pixel is given class s_1 and the right pixel receives class s_2. The same holds for any pair of vertically adjacent pixels and ordered pairs of classes that do not belong to V. An adjacency pattern specification and a corresponding image segmentation is illustrated in Figure 5.2.

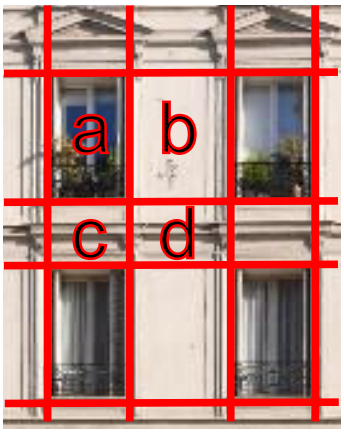
For any grid pattern shape prior, $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{H}, \mathcal{V})$ there exists an adjacency pattern $A = (S, V, H)$ encoding the same set of shapes. In the following, we construct an adjacency pattern equivalent to a given grid pattern. We define the set of pre-semantic pixel classes as $S = \mathcal{R} \times \mathcal{C}$. In consequence, the sets of classes assigned to image pixels are the same for both types of priors. We define the sets V and H in such a way to enforce the same structure on the possible labelings as the one resulting from the grid pattern formulation. We enforce that the rows of a labeling conforming to the adjacency pattern are valid rows of the grid pattern by requiring that each two horizontally adjacent pixels receive classes with the same row-class



$$\begin{aligned}
\mathcal{R} &= \{A, B\},\\
\mathcal{C} &= \{1, 2\},\\
\Psi(A, 1) &= \text{window},\\
\Psi(A, 2) &= \text{wall},\\
\Psi(B, 1) &= \text{wall},\\
\Psi(B, 2) &= \text{wall},\\
\mathcal{V} &= \{(A, A), (B, B), (A, B), (B, A)\},\\
\mathcal{H} &= \{(1, 1), (2, 2), (1, 2), (2, 1)\}.
\end{aligned}$$

Figure 5.1: *Left*: grid-shaped segmentation, where row and column labels induce a pixel labeling. *Right*: specification of the corresponding grid pattern using row and column classes.

$$\begin{aligned}
(A, 1) &= a,\\
(A, 2) &= b,\\
(B, 1) &= c,\\
(B, 2) &= d.
\end{aligned}$$



horiz. neighbors

h ↱	a	b	c	d
a	+	+	−	−
b	+	+	−	−
c	−	−	+	+
d	−	−	+	+

vert. neighbors

v ↱	a	b	c	d
a	+	−	+	−
b	−	+	−	+
c	+	−	+	−
d	−	+	−	+

Figure 5.2: *Left*: the mapping from pairs of row and column classes, defined in figure 5.1, to pixel classes. *Middle-left*: the segmentation from figure 5.1 labeled in terms of pixel classes. *Right*: Specification of the grid pattern using pairs of classes that can be assigned to vertically and horizontally adjacent pixels ('+' denotes an allowed adjacency, '–' a forbidden one).

component. We do the same thing for vertically adjacent pixels and the column-class component of pixel classes. We reformulate the constraints on the allowed classes of adjacent rows in terms of the row-class components of classes assigned to each two vertically adjacent pixels. We denote the row-class component of a pixel class $s = (r, c)$ by $r_s = r$ and its column-class component by $c_s = c$. The sets of allowed classes of adjacent pixels are defined as follows:

$$V = \{(s_1, s_2) | c_{s_1} = c_{s_2} \wedge (r_{s_1}, r_{s_2}) \in \mathcal{V}\} \quad \text{and} \tag{5.1a}$$

$$H = \{(s_1, s_2) | r_{s_1} = r_{s_2} \wedge (c_{s_1}, c_{s_2}) \in \mathcal{H}\} \ . \tag{5.1b}$$

We state the equivalence of the two priors formally below.

Lemma 2. *Consider a shape prior encoded as a grid pattern $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{H}, \mathcal{V})$, as defined in section 5.3.1. An adjacency pattern $A = (S, V, H)$, where $S = \mathcal{C} \times \mathcal{R}$ and V and H are defined according to* (5.1)*, encodes a prior that is equivalent to the grid pattern. That is, the set of image labelings that conform to the grid pattern $\mathcal{G}$ is the same as the set of labelings that conform to the adjacency pattern A.*

Proof. First we prove that if a segmentation is consistent with the grid pattern it is also consistent with the adjacency pattern. We need to show that, in a segmentation consistent with the grid pattern, classes of pairs of vertically and horizontally adjacent pixels belong to V and H, respectively. This follows directly from (5.1).

Next, we show that if a segmentation is not consistent with the grid pattern, then it is necessarily not consistent with the adjacency pattern. There are two possible ways in which a segmentation can be inconsistent with a grid pattern: (1) a row of the segmentation contains two pixels labeled (r, c) and (r', c') where $r \neq r'$, or (2) row classes r and r' are assigned to neighboring rows, while $(r, r') \notin \mathcal{V}$. Analogical conditions can be presented for columns. Condition (1) implies that there exists a sequence of horizontally adjacent pixels, labeled $s_1, \ldots s_n$, where $s_1 = (r, c)$ and $s_n = (r', c')$. In consequence there is at least one pair of horizontally adjacent pixels in the sequence that receive classes $s_a = (r_a, c_a)$ and $s_b = (r_b, c_b)$, such that $r_a \neq r_b$ and, according to (5.1), $(s_a, s_b) \notin H$. Therefore the segmentation is not consistent with the adjacency pattern. Condition (2) implies that there exists a pair of vertically neighboring pixels, labeled $s_1 = (r_1, c_1)$ and $s_2 = (r_2, c_2)$ such that $(r_1, r_2) \notin \mathcal{V}$, and therefore $(s_1, s_2) \notin V$ according to (5.1) and the segmentation is inconsistent with the adjacency pattern. □

In practice, the number of allowed adjacencies is small: $|V|, |H| \ll |S|^2$. If necessary, the tables V, H can thus be encoded as sparse matrices. Note that these vertical and horizontal adjacencies actually correspond to the transitions of two non-deterministic finite automatons, defining languages of strings of pixel labels that can occur on a column or row of pixels. However, the two automatons are coupled in that all rows and columns must agree, i.e., neighboring pixel compatibility has to be valid both vertically and horizontally.

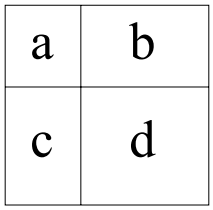
Adjacency patterns happen to be more general than grid patterns in that they can model more complex segmentations (see Section 5.3.2) as well as occlusions (see Section 5.5).

a	b	a	b
c	d	c	d
a	b	a	b
c	d	c	d

h↱	a	b	c	d
a	+	+	−	−
b	+	+	−	−
c	−	−	+	+
d	−	−	+	+

v↱	a	b	c	d
a	+	−	+	−
b	−	+	−	+
c	+	−	+	−
d	−	+	−	+

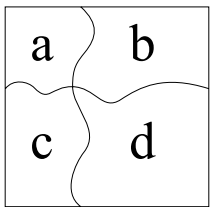
(a) Repetition pattern on grid with straight axis-aligned boundaries



h↱	a	b	c	d
a	+	+	−	−
b	−	+	−	−
c	−	−	+	+
d	−	−	−	+

v↱	a	b	c	d
a	+	−	−	−
b	−	+	−	−
c	+	−	+	−
d	−	+	−	+

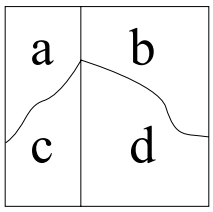
(b) Fixed pattern on grid with straight axis-aligned boundaries



h↱	a	b	c	d
a	+	+	+	−
b	−	+	−	+
c	+	−	+	+
d	−	+	−	+

v↱	a	b	c	d
a	+	+	−	−
b	+	+	−	−
c	+	−	+	+
d	−	+	+	+

(c) Fixed pattern on grid with winding axis-driven boundaries



h↱	a	b	c	d
a	+	+	+	−
b	−	+	−	−
c	−	−	+	+
d	−	−	−	+

v↱	a	b	c	d
a	+	−	−	−
b	−	+	−	−
c	+	−	+	−
d	−	+	−	+

(d) Fixed pattern on grid with monotonous boundaries

Figure 5.3: Examples of shape patterns and corresponding horizontal and vertical compatibility tables for neighboring pixel classes: '+' denotes a pair of allowed neighbors in this order, '–' denotes forbidden pairs.

5.3.2 Handling complex patterns and boundaries

In real images, the boundaries between some semantic classes, like 'roof' and 'sky', are often irregular and cannot be modeled by straight axis-aligned line segments. A wide class of priors expressing patterns with such complex boundaries can be encoded in terms of an adjacency pattern by properly designing the sets of allowed neighbor classes, V and H. Several examples are given in Figure 5.3. They can be thought of as an alphabet of basic adjacency patterns.

Figure 5.3a recalls the pattern already shown in Figure 5.1: it is a repetition pattern on a grid with straight, axis-aligned boundaries. The height and width of rows and columns can vary. It can be used to model, e.g., a grid of windows on a background of wall.

Figure 5.3b represents a fixed pattern on a grid with straight axis-aligned boundaries. The difference with respect to the previous case is that here the prior does

not allow for repetition of the alternation of classes. For example, looking at the pattern from left to right, a transition of class 'a' to 'b' is possible, but a transition from 'b' to 'a' is not.

As shown on Figure 5.3c, these straight borders can be turned into irregular winding boundaries by allowing a controlled interpenetration of classes. For instance, on a horizontal line, an 'a' can now be followed by a 'c' and then again by an 'a', but a 'b' on this line still cannot be preceded by 'c'.

Figure 5.3d displays another variant where monotonicity is imposed to a boundary, to represent a rising and a descending border. Such a pattern can be used to model a roof, which is expected to have an ascending slope in the beginning and a descending slope at the end. It can also be used to model a chimney on a roof.

Note that, when used for encoding priors for patterns with non-linear, winding boundaries, adjacency patterns are limited to patterns where symmetric pairs of classes cannot be assigned to neighboring pixels. Allowing both (a, b) and (b, a) to be classes of vertically, or horizontally neighboring cells, with a winding boundary in between, would result in the sets V and H containing all possible configurations of these two classes. Such an adjacency pattern would then merely express any combination of a and b. However, this limitation does not substantially constrain the scope of applications of the proposed model, as the irregular boundaries do not form full grids in facade images (for example, the sky-roof boundary appears only once in a facade).

5.3.3 General Remarks on Adjacency Patterns

Adjacency patterns can easily be extended to constrain classes that appear at the boundary of the whole image, for example, to impose that a properly cropped facade can start with 'wall' or 'sky', but not 'window'. Priors modeling regions of minimum height or width m can also be defined by introducing classes $a_1, \ldots, a_m$ and forbidding all adjacencies but from a_i to a_{i+1}. Similarly, a maximum height or width can be imposed by also allowing adjacencies from any a_i to any class that can follow a_m. However, the number of required classes with this encoding increases linearly with the minimum or maximum size.

Theoretically it is possible to construct adjacency patterns (and grid patterns as well) expressing priors for which there is no conforming labeling of an image of particular size. A trivial example is $H = V = \emptyset$, in which case only labelings of size one by one pixel can be consistent with the adjacency pattern. However, this aspect is of limited practical importance. It is enough that there exists a class $s \in S$ such that $(s, s) \in V$ and $(s, s) \in H$ to guarantee existence of a labeling consistent with the adjacency pattern (where all pixels are labeled s), independently of image size.

5.3.4 Soft penalties on pairs of classes of adjacent pixels

It may also be beneficial to consider a pairwise term penalizing frequent transitions between classes, to limit noise in the resulting segmentations and to express soft preference for some segmentations over the others. We denote this pairwise potential by $\theta : S \times S \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. The potentials can be learned

using the structured learning approach [25]. In our experiments we assume the Potts form of the pairwise potentials and we set the penalty for assigning different classes to neighboring pixels using grid search.

5.4 Hierarchical adjacency patterns

A formalism of shape priors encoding a single grid pattern would not feature a practical-enough expressive power for the task of facade parsing. The layout of facade elements is usually more complex than a grid. Encoding patterns containing multiple misalignments by means of a single grid requires a number of pixel classes that grows exponentially with the number of vertically and horizontally misaligned objects. This is because each transition between semantic classes, that impacts the semantic labeling only locally, has to extend through the whole image, 'cutting' objects located in other parts of the image into subrectangles of different pre-semantic classes. This effect is illustrated in the left part of figure 5.4.

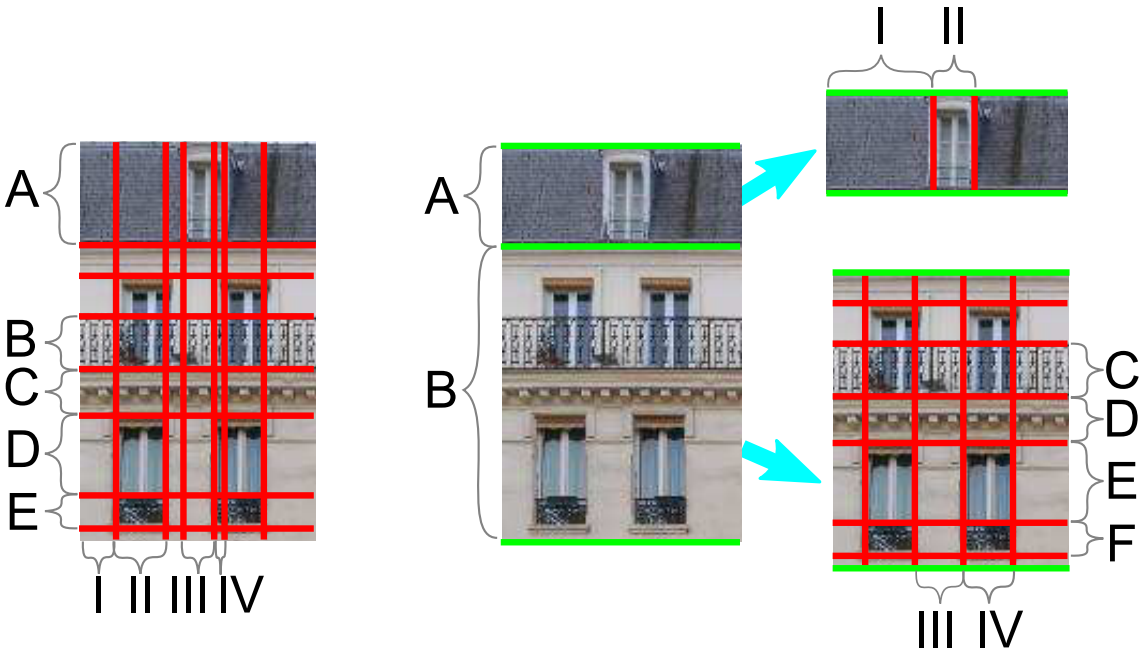


Figure 5.4: *Left*: modeling a pattern with vertical misalignment as a single grid requires each column class to encode the type of both the element occupying the lower part of the column and the element occupying its upper part: *I≈(wall, roof)*, *II≈(window, roof)*, *III≈(wall, attic window)*, *IV≈(window, attic window)*. The number of resulting pixel classes (20 in the depicted case) is exponential in the number of misalignments. *Right*: a hierarchical grid model, where cells of a coarser grid (green) are further subdivided into finer grids (red), results in a set of terminal pixel classes of cardinality (10 in the example) linear in the number of misalignments.

To address this issue, one could define a hierarchy of nested grids, where a cell of a coarse grid can be further subdivided into a finer grid. We propose to model complex shapes using a hierarchy of adjacency patterns. Transition between levels of the hierarchy is realized by mapping pre-semantic pixel classes of an adjacency pattern on a coarser level of the hierarchy to other adjacency patterns on a finer level. The concept is that a connected region of pixels that received the same pixel

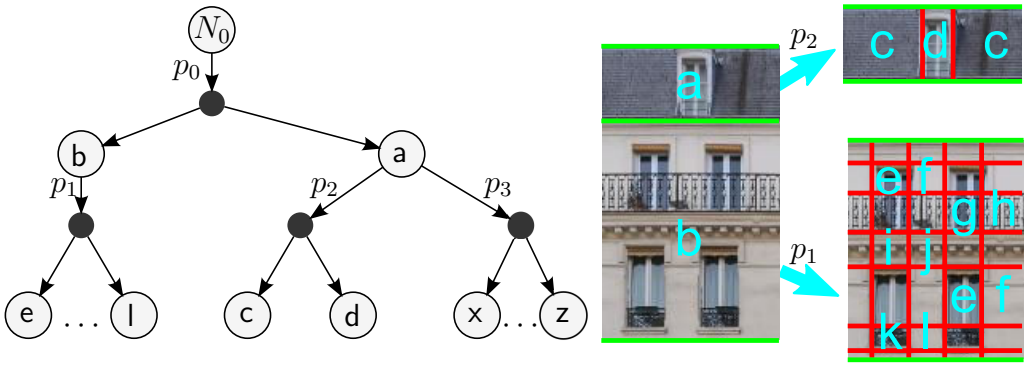


Figure 5.5: A hierarchy of adjacency patterns and a segmentation conforming to the prior encoded by the hierarchy. Large, circled nodes correspond to pixel classes. Small, filled nodes correspond to adjacency patterns. Productions are marked next to arrows that represent the mapping from pixel classes to adjacency patterns. The initial nonterminal N_0 is omitted in the illustration on the right. Note that the hierarchy on the left encodes a structural alternative. The roof a can be developed in two alternative ways, by production p_2 or production p_3. Only an application of p_2 is pictured on the right. The grid pattern corresponding to p_3, is not represented here.

class can be further segmented using a prior encoded by the adjacency pattern to which the class is mapped. We discriminate between nonterminal and terminal pixel classes, the first of which are mapped to adjacency patterns at a finer scale, and the second terminate the hierarchy. Only the terminal classes are present in the final segmentation. A hierarchical adjacency pattern is a quadruple $\hat{A} = (\mathcal{N}, \mathcal{T}, N_0, \mathcal{P})$ where $\mathcal{N}$ is a finite set of nonterminal classes, $\mathcal{T}$ is a finite set of terminal classes, disjoint from $\mathcal{N}$, $N_0 \in \mathcal{N}$ is the start symbol and $\mathcal{P}$ is a set of production rules of the form $p = N_p \to A_p$ where $N_p \in \mathcal{N}$ and $A_p = (S_p, V_p, H_p)$ is an adjacency pattern such that $S_p \subset \mathcal{N} \cup \mathcal{T}$. Additionally, we impose that the productions contain no cycle and that the sets of pixel classes in each adjacency pattern A_p are all disjoint. More formally,

$$\forall s \in \mathcal{T} \cup \mathcal{N} \setminus \{N_0\}, \exists! p \in \mathcal{P} \text{ such that } s \in S_p \; . \tag{5.2}$$

The concept of the hierarchy is illustrated in the right part of figure 5.5.

In order to formalize the notion of conformance of a labeling to a hierarchical adjacency pattern $\hat{A}$, we first introduce the concept of ancestor-descendant relation in the hierarchy. We say that a pixel class a descends, or is derived from a nonterminal pixel class b on a coarser level of the hierarchy, if a belongs to an adjacency pattern which is used to subdivide a region of class b. The notion can be extended to relations across many levels of the hierarchy. We define the set of descendants of a terminal class as a singleton set containing this class. The set of descendants of a nonterminal class N contains terminal classes of the adjacency patterns to which the nonterminal is mapped, and descendants of all the nonterminals of these adjacency patterns. We recall that productions $p \in \mathcal{P}$ are of the form $p = N_p \to (S_p, V_p, H_p)$. Formally, the set of descendants of class s is defined as

$$Desc(s) = \begin{cases} \{s\} & \text{if } s \in \mathcal{T} \\ \bigcup_{p \in \mathcal{P} \text{ s.t. } N_p = s} \left((S_p \cap \mathcal{T}) \cup \bigcup_{s' \in S_p \cap \mathcal{N}} Desc(s') \right) & \text{if } s \in \mathcal{N} \end{cases} . \tag{5.3}$$

The recursion in the above definition always terminates, because according to (5.2) there are no cycles in the hierarchy of adjacency patterns. Similarly, we can define the descendants of a production $p \in \mathcal{P}$ as

$$Desc(p) = \bigcup_{s \in S_p} Desc(s) \ . \tag{5.4}$$

A class s that descends from a production p necessarily has an ancestor class in the adjacency pattern of that production, denoted $Anc_p(s)$

$$Anc_p(s) = s' \text{ s.t. } s' \in S_p \wedge s \in Desc(s') \ . \tag{5.5}$$

We now formalize the notion of conformance of a segmentation to a hierarchical adjacency pattern. We require that for each production p, each region of the labeling that contains only classes descending from that production $s \in Desc(p)$, conforms to the adjacency pattern of that production A_p. We denote the set of pixels, represented as pairs (i, j), where i indexes an image row and j indexes an image column, by $\mathcal{I}$. The set of pairs of pixel indexes excluding the last column is denoted by $\mathcal{I}_h$, and the set of pairs of row and column indexes without the last row by $\mathcal{I}_v$. We denote the class assigned to pixel $(i, j) \in \mathcal{I}$ as s_{ij}. The conformance conditions can be formulated as

$$\forall (i,j) \in \mathcal{I}_h,\ \forall p \in \mathcal{P},\ s_{ij}, s_{ij+1} \in Desc(p) \implies \big(Anc_p(s_{ij}), Anc_p(s_{ij+1})\big) \in H_p \ , \tag{5.6a}$$

$$\forall (i,j) \in \mathcal{I}_v,\ \forall p \in \mathcal{P},\ s_{ij}, s_{i+1\,j} \in Desc(p) \implies \big(Anc_p(s_{ij}), Anc_p(s_{i+1\,j})\big) \in V_p \ . \tag{5.6b}$$

A hierarchical adjacency pattern can be represented as a simple adjacency pattern over the terminal classes of the hierarchical adjacency pattern. The hierarchical representation is more convenient when the pattern is specified by a human user, because it requires defining a lower number of constraints on the classes of adjacent pixels. Two matrices of allowed neighboring classes need to be defined for each adjacency pattern in the hierarchy, but the number of classes in each of the patterns is normally low. To the contrary, a single adjacency pattern encoding the prior requires specifying matrices of the size of number of terminal classes squared.

Given $\hat{A}{=}(\mathcal{N}, \mathcal{T}, N_0, \mathcal{P})$, a hierarchical adjacency pattern, we now explain how to transform it into an equivalent, flattened adjacency pattern $A = (S, V, H)$. The set of classes of the flattened adjacency pattern is equal to the set of terminal classes of the hierarchical pattern

$$S = \mathcal{T} \ . \tag{5.7}$$

The definition of the sets of pairs of classes that can be assigned to vertically and horizontally adjacent pixels, V and H, follows directly from the conformance conditions (5.6).

$$V = \Big\{ (s_1, s_2) \in \mathcal{T}^2 \mid \forall p \in \mathcal{P} \text{ s.t. } s_1, s_2 \in Desc(p) \\ \big(Anc_p(s_1), Anc_p(s_2)\big) \in V_p \Big\} \tag{5.8a}$$

$$H = \Big\{(s_1, s_2) \in \mathcal{T}^2 \mid \forall p \in \mathcal{P} \text{ s.t. } s_1, s_2 \in Desc(p) \\ \big(Anc_p(s_1), Anc_p(s_2)\big) \in H_p\Big\} . \tag{5.8b}$$

Representing the shape prior of all the presented types, including one encoded by a hierarchical adjacency pattern, in the form of a flattened adjacency pattern $A = (S, V, H)$ enables formulating the inference in terms of the MAP-MRF problem, as shown in section 5.7.

5.5 Handling occlusions

Occlusions are omnipresent in urban scenes. For facade parsing, the most common occlusions are by trees and lamp posts. Lower parts of facades can also be occluded by other types of vegetation, street signs, cars and pedestrians. Vegetation can be found on balconies and window ledges too.

Given an adjacency pattern $A = (S, V, H)$, we consider a set of semantic classes of occluding objects O, disjoint from the set of pre-semantic classes S and from the set of semantic classes of facade elements K. A pixel representing a non-occluded part of the facade should be assigned the semantic class $k \in K$ determined by the mapping Ψ from its 'pre-semantic' class $s \in S$. In case of occlusion, the pixel should be assigned the class of the occluding object $o \in O$, which cannot be determined from its 'pre-semantic' class. However, we observe that only a small number of combinations of occluder and pre-semantic classes is semantically meaningful. For example, pedestrians and cars can only occlude the lower part of a facade, but not the roof. We represent the semantically meaningful pairs of classes by a set $\mathfrak{S} \subset S \times O$.

A prior of hierarchical grid patterns with occlusions can be formulated in terms of allowed transitions between classes of vertically and horizontally neighboring pixels, by means of an adjacency pattern $A_o = (S_o, V_o, H_o)$ defined as follows. Each pixel class $\sigma \in S_o$ has a 'pre-semantic' and a 'semantic' component $\sigma = (s, \kappa)$, where $s \in S$ and $\kappa \in (O \cup K)$. The set $S_o \subset S \times (O \cup K)$ consists of semantically consistent pairs, i.e., such that either the pixel is not occluded and then $\kappa = \Psi(s)$, or the pixel is occluded and then $(s, \kappa) \in \mathfrak{S}$. More formally,

$$S_o = \{(s, \Psi(s)) | s \in S\} \cup \mathfrak{S} . \tag{5.9}$$

This practically limits the number of classes. In our experiments, the number of classes never increased by a factor of more than 2.5, compared to the model without occlusions. The sets V_o and H_o of ordered pairs of classes that can be assigned to vertically and horizontally adjacent pixels are determined by the pre-semantic components of the classes. We define the pre-semantic components of a class $\sigma \in S_o$, $\sigma = (s_\sigma, \kappa_\sigma)$, as $s(\sigma) = s_\sigma$. The sets of classes that can be defined to adjacent pixels can be defined as

$$V_o = \Big\{(\sigma_1, \sigma_2) | \sigma_1, \sigma_2 \in S_o, \big(s(\sigma_1), s(\sigma_2)\big) \in V\Big\} , \tag{5.10a}$$

$$H_o = \Big\{(\sigma_1, \sigma_2) | \sigma_1, \sigma_2 \in S_o, \big(s(\sigma_1), s(\sigma_2)\big) \in H\Big\} . \tag{5.10b}$$

The pairwise potential for an adjacency pattern with occlusions is denoted $\theta_o : S_o \times S_o \rightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. The mapping of pixel classes to semantic or occluder classes for the pattern with occlusion becomes $\Psi_o(\sigma, \kappa) = \kappa$.

5.6 Relation to Split Grammars

In this section, we discuss the expressive power of split grammars, as applied to facade parsing by Teboul et al., [60, 59], and the expressive power of adjacency patterns. Adjacency patterns can encode priors expressing hierarchical rectangular tilings. By a hierarchical rectangular tiling we understand a segmentation that can be obtained by subdividing the initial image into a grid and then, recursively, subdividing each of the grid cells in a similar fashion. Split grammars were introduced in more detail in section 2.1.2. From a theoretical point of view, they can be considered context-free grammars, and therefore their expressive power is larger than that of adjacency patterns. For example, a context free grammar can encode structural symmetries (for example a recursion $A \rightarrow BAB$ produces patterns of type $BBABB$, with equal number of Bs on both sides of A). Encoding such segmentations in terms of adjacency patterns requires auxiliary pre-semantic classes for encoding the depth of recursion, which makes it impractical for a large number of B's and which is limited to the case where the number of B's is bounded. However, in practice, the split grammars that were successfully used for image parsing actually generate regular languages rather than context-free languages and thus, they can be represented as adjacency patterns. As shown in the experiments section, our priors actually encode much more complex patterns than the ones shown to work with existing split grammar frameworks. This is due to the fact that our formulation lends itself to an optimization scheme that is more effective than a randomized exploration of the space of grammar alternatives, used with split grammars. The proposed optimization algorithm is presented in section 5.8.

The split grammar parsing framework [60, 59] enables encoding the size of facade elements. Encoding size constraints using adjacency patterns requires an impractically large number of pre-semantic classes, equal to the size of the object in pixels.

Although we do not encode absolute element sizes in the framework of adjacency patterns, we can address the problem of frequent transitions between classes, resulting from noisy pixel classification, by penalizing the transitions with a positive cost. The mechanism can be used not only to prevent segments of small sizes, but also to express preference on some facade patterns over the others.

We also demonstrated that, under mild assumptions, adjacency patterns can model priors expressing segmentations where the boundaries between elements can take irregular shapes (as opposed to axis aligned straight line segments). Such a mechanism was never demonstrated to work with split grammars.

5.7 Optimal segmentation model

In this section we propose a formulation of the optimal image segmentation that conforms to an adjacency pattern. We denote image height and width by h and

w, the set of image row indexes $I = \{1, \dots h\}$, the set of image column indexes $J = \{1, \dots w\}$, and the image by $\mathcal{I} = I \times J$. We encode the assignment of a class $\sigma \in S_o$ to each pixel $(i, j) \in \mathcal{I}$ by variables $z_{ij\sigma} \in \{0, 1\}$, where $z_{ij\sigma} = 1$ if σ is the class assigned to pixel (i, j) and $z_{ij\sigma} = 0$ otherwise. To enforce the satisfaction of the constraints on classes of neighboring cells, we also introduce variables $v_{ij\sigma\sigma'} \in \{0, 1\}$ and $u_{ij\sigma\sigma'} \in \{0, 1\}$, modeling assignment of pairs of classes to pairs of vertically and horizontally neighboring pixels. More precisely, $u_{ij\sigma\sigma'} = 1$ if pixel (i, j) is assigned class σ and pixel $(i, j + 1)$ is assigned class σ', and $u_{ij\sigma\sigma'} = 0$ otherwise. The variables $v_{ij\sigma\sigma'}$ have similar interpretation and encode assignment of pairs of classes to vertically neighboring pixels. We denote the vectors of all $z_{ij\sigma}$, $u_{ij\sigma\sigma'}$, $v_{ij\sigma\sigma'}$ as $\mathbf{z}$, $\mathbf{u}$, $\mathbf{v}$, respectively. The goal is to find an assignment that minimizes the sum of costs $\phi_{ij\kappa}$ of assigning a class $\kappa \in O \cup K$ to each pixel $(i, j) \in \mathcal{I}$, and the costs θ_o of assigning pairs of classes to pairs of adjacent pixels, and satisfies the hard constraints on classes of adjacent pixels. We denote the set of all pixels except for the last row by $\mathcal{I}_v = (I \setminus \{h\}) \times J$, and the set of all pixels without the last column by $\mathcal{I}_h = I \times (J \setminus \{w\})$. The energy minimization takes the form

$$\min_{\mathbf{z},\mathbf{v},\mathbf{u}} \sum_{\substack{(i,j)\in\mathcal{I} \\ \sigma\in S_o}} \phi_{ij\Psi_o(\sigma)} z_{ij\sigma} + \sum_{\substack{(i,j)\in\mathcal{I}_v \\ \sigma,\sigma'\in S_o}} \theta_o(\sigma, \sigma')\, v_{ij\sigma\sigma'} + \sum_{\substack{(i,j)\in\mathcal{I}_h \\ \sigma,\sigma'\in S_o}} \theta_o(\sigma, \sigma')\, u_{ij\sigma\sigma'}. \tag{5.11}$$

We require that exactly one class is assigned to each pixel,

$$\forall (i, j) \in \mathcal{I}, \quad \sum_{\sigma\in S_o} z_{ij\sigma} = 1\ . \tag{5.12}$$

We impose consistency between variables encoding pixel labels and pairs of labels of adjacent pixels

$$\forall (i, j) \in \mathcal{I}_v,\ \forall \sigma \in S_o, \qquad \sum_{\sigma'\in S_o} v_{ij\sigma\sigma'} = z_{ij\sigma}, \qquad \sum_{\sigma'\in S_o} v_{ij\sigma'\sigma} = z_{i+1\,j\sigma'}, \tag{5.13}$$

$$\forall (i, j) \in \mathcal{I}_h,\ \forall \sigma \in S_o, \qquad \sum_{\sigma'\in S_o} u_{ij\sigma\sigma'} = z_{ij\sigma}, \qquad \sum_{\sigma'\in S_o} u_{ij\sigma'\sigma} = z_{ij+1\,\sigma'}. \tag{5.14}$$

We constrain the pairs of neighboring classes according to:

$$\forall (i, j) \in \mathcal{I}_v, \forall (\sigma, \sigma') \notin V_o, \quad v_{ij\sigma\sigma'} = 0\ , \tag{5.15a}$$

$$\forall (i, j) \in \mathcal{I}_h, \forall (\sigma, \sigma') \notin H_o, \quad u_{ij\sigma\sigma'} = 0\ . \tag{5.15b}$$

The presented model resembles a linear formulation of the most probable configuration of a Markov Random Field [68]. Although the presence of hard constraints on classes of neighboring pixels differentiates the problem from a standard MRF over a pixel grid, frequently used in computer vision, a number of algorithms used for solving the MAP-MRF problem could be applied to approximate the optimal solution to our problem.

5.8 Inference

To solve problem (5.11-5.15) we adopt the dual decomposition approach [26, 54], outlined in appendix B. We relax the constraints on binary domain of variables $z_{ij\sigma}$,

$u_{ij\sigma\sigma'}$ and $v_{ij\sigma\sigma'}$, obtaining a large linear program. To solve it efficiently, we transform the original objective to a sum of easy to optimize functions and formulate a dual problem that can be solved by iteratively solving the simple objectives. We adopt the most standard decomposition of a 4-connected grid into Markov chains over image rows and columns. The resulting subproblems can be solved independently and efficiently using the Viterbi algorithm. Our implementation of the Viterbi algorithm exploits the hard constraints on classes of adjacent pixels to obtain a significant speedup over the generic version of the algorithm. From the solution to the dual problem, we extract a primal, binary solution that satisfies the hard constraints. We present a detailed derivation of the complete inference algorithm in appendix D.

5.9 Experiments

We evaluated the accuracy of our algorithm in segmenting facade images on a wide range of datasets and for unary terms of various quality. We emphasize that our goal is not to establish a new state-of-the-art performance by using more accurate classification algorithms, better features, or better detections. Instead we demonstrate that the proposed optimization scheme leads to better segmentations given the same bottom-up cues. Moreover, we show that imposing the structural constraints improves parsing results, while previous work [33] suggested that structural correctness comes at a cost of decreased accuracy.

Convergence and duality gap. The algorithm operates on the dual problem, yielding a lower bound on the optimal energy. The gap between the dual energy and the energy of the primal binary solution can be seen as a measure of suboptimality of the obtained solution. We analyze the performance of the algorithm on the ECP dataset [59] against the ground truth produced by Martinović et al. [33]. For each image of the test set, we record the dual energy in each iteration of the algorithm. We normalize the dual energies with respect to the energy of the final primal solution. We present the statistics in figure 5.6. For a vast majority of the images, the primal-dual gap is not more than 0.2% of the final energy. To verify to what percentage of mislabeled pixels the number corresponds, we performed an experiment in which we randomly relabeled a fraction of pixels in the solution obtained by our algorithm and observe the resulting change in the energy. The result of our experiment indicates that an increase of the final energy by 0.2% roughly corresponds to randomly flipping the labels of 0.2% pixels.

Performance on the ECP dataset. We apply our method to the ECP dataset [59], consisting of 104 images of Haussmannian building facades. We use the ground truth annotations proposed by Martinović et al. [33]. To obtain the per-pixel energies we apply the procedure described in Cohen et al. [9]: a multi-feature extension of TextonBoost implemented by Ladický et al. [29]. We describe the procedure in section 3.7. Due to smoothness of the texture classification, in this experiment we do not use the pairwise potentials. We use a highly flexible prior, where balconies can begin and end in any position along the facade (but need to enclose at least

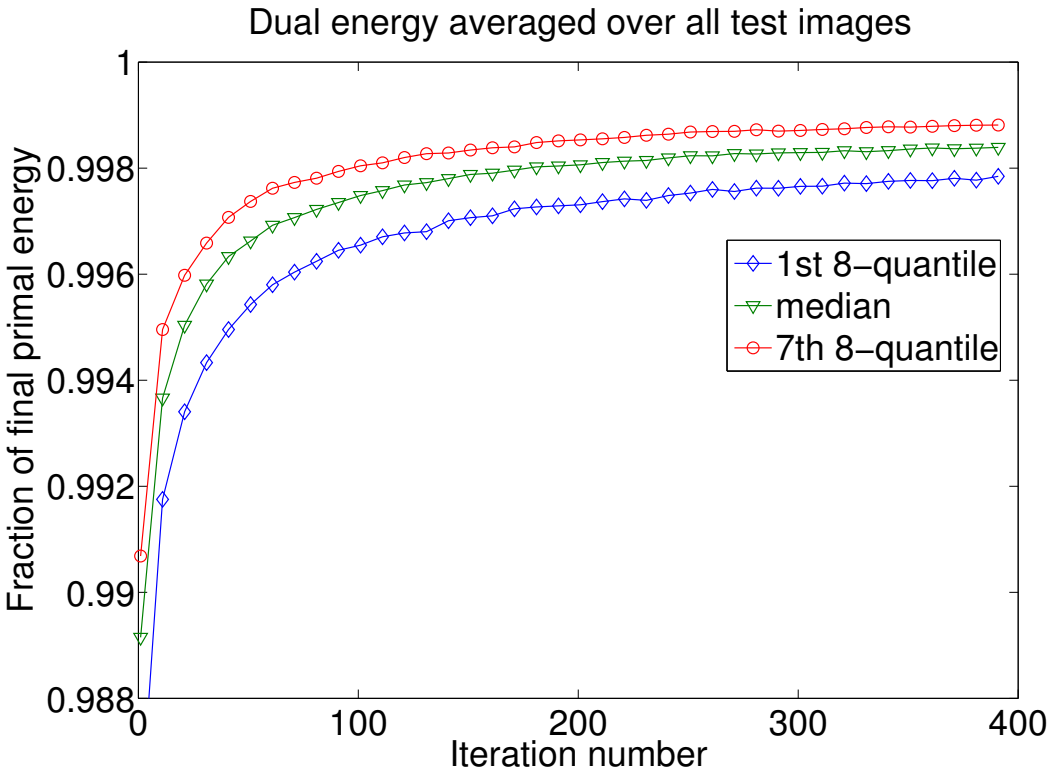


Figure 5.6: Statistics of the ratio of dual energy to the final primal energy with respect to iteration number. Experiment performed on the ECP dataset.

one window). The balconies can also have two different heights, corresponding to the small barriers usually limited to a single window and larger balconies. We use irregular boundaries to model ascending and descending parts of the roof, at the ends of the facade. The windows in the topmost floor can be misaligned vertically with the rest of the facade, in which case they are aligned with the attic windows. The used adjacency pattern has about 80 pre-semantic classes. In figure 5.7, we show an image of the ECP dataset, its segmentation into pre-semantic classes and the segmentation after mapping the pre-semantic classes to semantic classes. Qualitative results of the experiment are illustrated in figure 5.8.

As shown in table 5.3, we outperform by a small margin state-of-the-art methods that use the same unaries. Additionally, our algorithm can accept user-defined shape priors, while [9] has hard-coded constraints. An advantage over the row and column label-based (RCL) formulation presented in chapter 4 comes from a more precise prior, modeling for example the irregular roof boundaries and different balcony heights on the same floor. We also outperform the RCL method in terms of running time: 100 iterations of algorithm 5 takes less than 30 seconds (with a CPU implementation running on a 3GHz Core-i7 processor), compared to 4 minutes in the latter case.

For a fair comparison with the results of the work by Martinović et al. [33], we perform another experiment on the ECP dataset using the same bottom-up cues as in their paper (kindly made available by the authors): the output of a Recursive Neural Network [53], which is less accurate than TextonBoost. The testing protocol

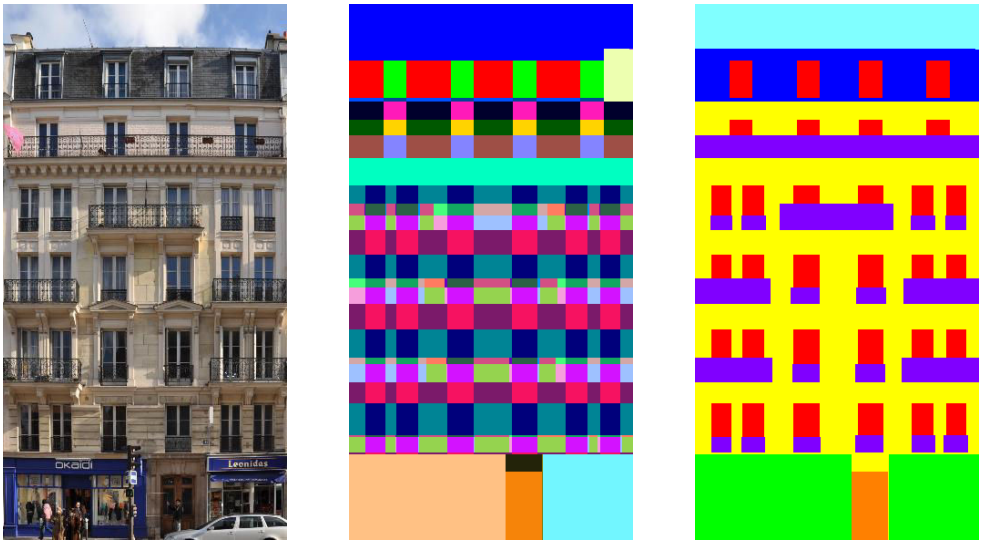


Figure 5.7: An image of the ECP dataset [59] (left), its segmentation into pre-semantic (middle) and semantic classes (right). The complex pattern of windows and balconies exemplifies the high expressive power of the proposed prior formalism.

Table 5.3: Performance on the ECP dataset with two kinds of unary potentials: using a Recursive Neural Network [53] and using a variant of TextonBoost [29]. The rows corresponding to classes present class accuracy. The bottom rows contain average class accuracy and total pixel accuracy. In columns, starting from left: performance of the raw RNN classifier; result of [33]; result of the graph grammar parsing algorithm (GG) presented in chapter 3, using the same unaries and window detections; results of the row and column labeling (RCL) algorithm presented in chapter 4, with the same unaries; our result for the same unaries with adjacency patterns (AP); performance resulting from classifying each pixel independently using the TextonBoost scores; results of Cohen et al. [9]; results of the graph grammar parsing algorithm (GG) of chapter 3 with the same unaries and window detections, as described in chapter 3; results of algorithm for row and column labeling (RCL) presented in chapter 4; results of the proposed adjacency pattern-based formulation (AP).

	RNN unaries					TextonBoost unaries				
	raw	[33]	GG	RCL	AP	raw	[9]	GG	RCL	AP
roof	70	74	85	83	78	89	90	85	91	91
shop	79	93	92	90	90	95	94	96	95	97
balcony	74	70	72	73	76	90	91	83	90	91
sky	91	97	94	92	94	94	97	96	96	97
window	62	75	70	57	67	86	85	77	85	87
door	43	67	0	40	44	77	79	0	74	79
wall	92	88	91	95	93	90	90	86	91	90
pixel accur.	82.6	84.2	84.8	85.5	**86.2**	90.1	90.8	85.5	90.8	**91.3**

is the same as used by Martinović et al. [33] and consists in creating five folds of a training, validation and test set. The first fold contains 60 training, 20 validation and 24 testing images. The remaining four folds contain 64 training, 20 validation and 20 testing images. For this experiment we use a simple pairwise Potts potential. We set the off-diagonal entries of pairwise cost tables to 0.5, a value determined by grid search on the validation data set. Qualitative results are presented in figure 5.8, and the resulting accuracies can be found in table 5.3. We outperform the baseline [33],

even though their segmentation is obtained using detections of windows, balconies and doors in addition to RNN. The influence of the detections on the performance of the baseline can be seen on results for the window and door class, for which the baseline outperforms our algorithm. Besides, our algorithm guarantees the semantic correctness of the segmentations, while the baseline aligns facade elements only locally and can yield, for example, balconies ending in the middle of a window.

Performance on the Graz50 dataset. The Graz50 dataset [47] contains 50 images of various architectural styles labelled with 4 classes. We compare the performance of our algorithm to the method of Riemenschneider et al. [47] and to the method based on row and column labels (RCL) presented in chapter 4. As in the case of the ECP dataset, we use the TextonBoost (TB) to get unaries. Again, we do not use the pairwise potentials due to smooth output of the texture classifier. We note that Riemenschneider et al. use a different kind of per-pixel energies, obtained by means of a random forest classifier. On the other hand the energies used in the row and column labeling (RCL) approach of chapter 4 are the same as in our algorithm. As shown in table 5.4, our algorithm outperforms the state of the art and yields shorter running times: less than 30 seconds per image compared to 4 minutes for the row and column label-based method. As the dataset does not contain irregular boundaries between facade elements, the increased accuracy can be attributed to a different formulation of the optimization problem, which is solved more efficiently.

Table 5.4: Results on the Graz50 dataset. The second column shows class-wise accuracy for results reported by Riemenschneider et al. [47]. The third column contains results of the row and column labeling approach (RCL) presented in chapter 4. The last column contains the results of the adjacency pattern (AP) formalism, presented in this chapter.

	Graz50		
	[47]	RCL	AP
sky	91	93	93
window	60	82	84
door	41	50	60
wall	84	96	96
average	69.0	80.3	83.5
pixel accur.	78.0	91.8	**92.5**

Performance on the ENPC dataset The ENPC dataset [16] consists of 80 images of facades of Art-Deco architectural style. We run the experiments on four folds of 20 testing and 60 training images. Again, we use the TextonBoost-based unary potentials. We use Potts form of pairwise potentials penalizing transitions between different classes with a fixed coefficient, determined by grid search on a subset of the training set. Many images of this dataset contain facades occluded by trees. The facade themselves feature more structural complexity than the ECP or Graz50 datasets. The results are presented in table 5.5 and figure 5.10. We use two

Figure 5.8: Example triplets of parsing results with adjacency patterns: original image (first of each three images), result of per-pixel classification (second image), parsing result (third image). The first two rows present results obtained using the TextonBoost potentials, the third and fourth rows contain results obtained using RNN. The last row contains examples of inaccurate segmentations (note that in the last segmentation there is a very thin line of windows between the two topmost balconies).

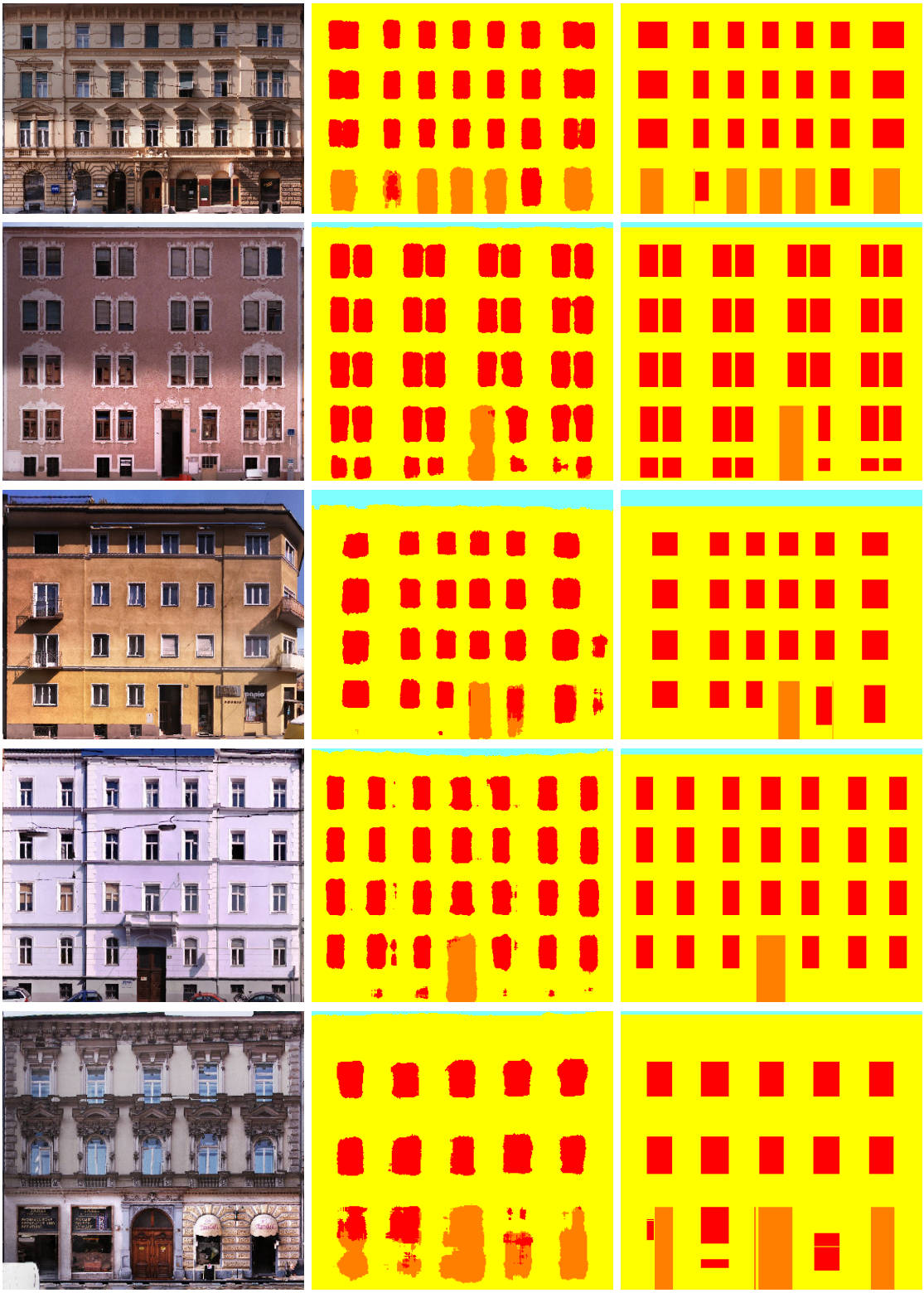


Figure 5.9: Example triplets of parsing results on Graz50 dataset [47] with TextonBoost potentials, using the adjacency pattern. The images in each triplet are arranged in the following order: original image, result of per-pixel classification, parsing result. The last row is an example of inaccurate segmentation.

Table 5.5: Results on the ENPC dataset. The first pair of columns contains accuracies resulting from classifying the pixels independently, for only the semantic classes (raw-s) and for both semantic and occluder classes (raw-o). The last two columns contain results of our parsing algorithm (both using the classifier with occlusions, 'raw-o'): the extracted facade structure (AP-struct) and the segmentation in terms of semantic and occluder classes (AP-occlud). The two results are evaluated using different, problem-specific variants of the ground truth.

	Art-Deco			
	raw-s	raw-o	AP-struct	AP-occlud
roof	82	82	81	82
shop	96	95	97	97
balcony	88	87	82	87
sky	97	97	98	97
window	87	85	82	82
door	64	63	57	57
wall	77	87	89	88
vegetation	–	90	–	90
pixel accur.	83.5	88.4	88.8	88.8

ground truth annotations, one that is oblivious of occlusions and only represents facade structure (even in image regions which represent the occluding tree), and another one, where occluding objects are segmented as well. When not modeling the appearance of occluding objects, the raw pixel classification in the occluded regions is poor, resulting in lower overall accuracy. This result is presented in the second column (raw-s) in table 5.5. Explicitly modeling occlusions enable recovering both the image segmentation, including the boundaries of the occluding object (the fourth column of the table, marked 'AP-occlud'), and the structure of the occluded facade (the third column of the table, marked 'AP-struct').

Performance on the eTrims dataset. We also tested our algorithm on the challenging eTrims dataset [27], consisting of 60 images of facades of different styles. We perform a 5-fold cross validation as in the experiments of Martinović et al. [33] and the experiments performed by Cohen et al. [9], and each time the dataset is divided into 40 training and 20 testing images (the test sets for different folds are not disjoint). We use per-pixel energies generated by a Recursive Neural Network, like in the work cited above. We assume the Potts model of pairwise potentials, with the parameter determined by grid search on a subset of the training set. The results are presented in table 5.6 and figure 5.11. Our algorithm outperforms the result of the three-layered method of Martinović et al. [33] and yields result only slightly inferior to the dynamic programming-based method of Cohen et al. [9]. The possible reason is the constraints assumed in the latter paper are less restrictive than our priors. However, our method is still the first algorithm with a user-specified shape prior to be tested on the eTrims dataset and its performance is a close match to the two baseline methods, which offer no flexibility with respect to prior definition.

Typical failures. A typical failure, that can be observed in the figures illustrating qualitative results, involves facade elements that are too small or too large along one

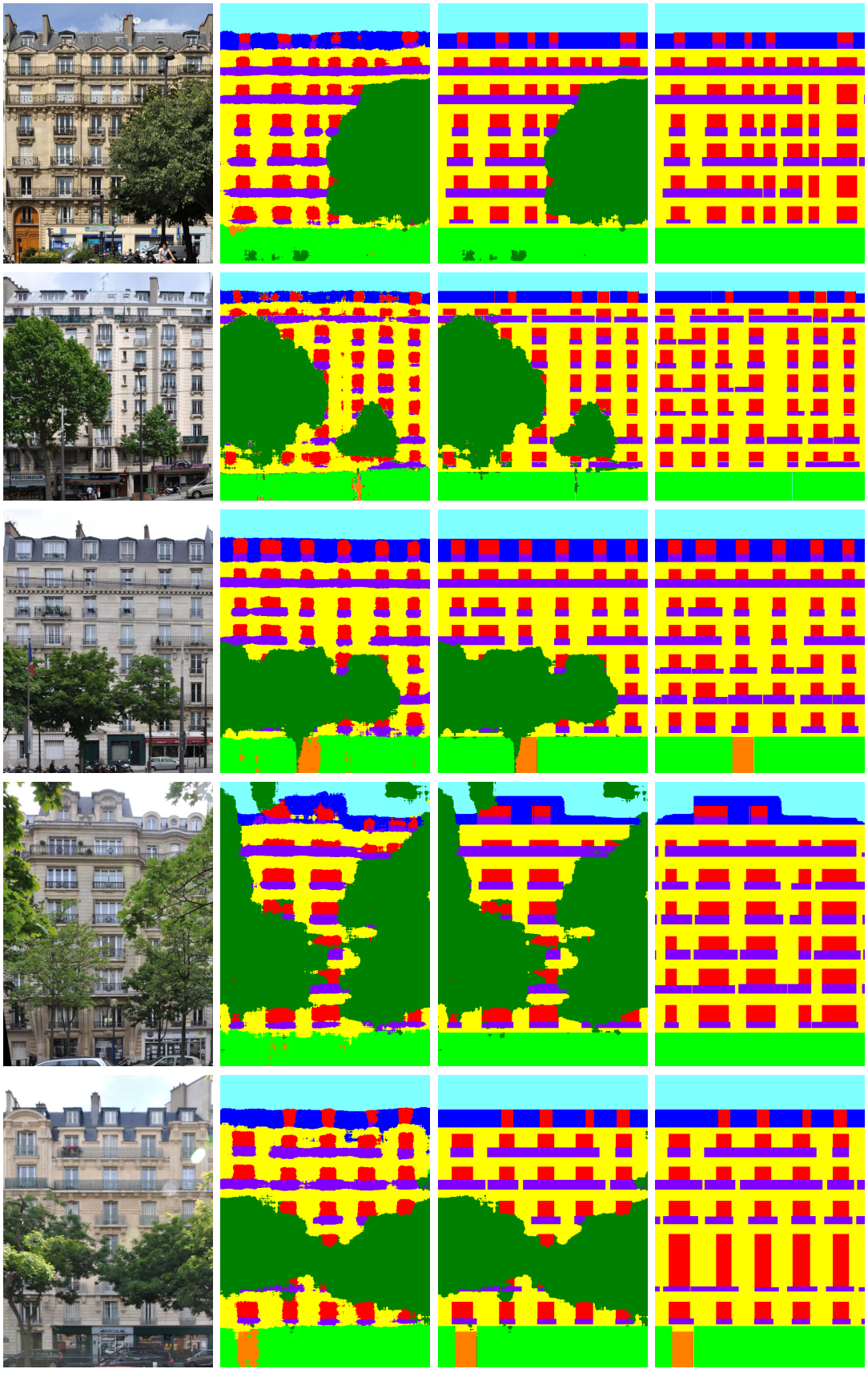


Figure 5.10: Results of parsing images with from the ENPC dataset. For each row, in the following order: original image, unary classification, segmentation with occluder classes, extracted facade structure.

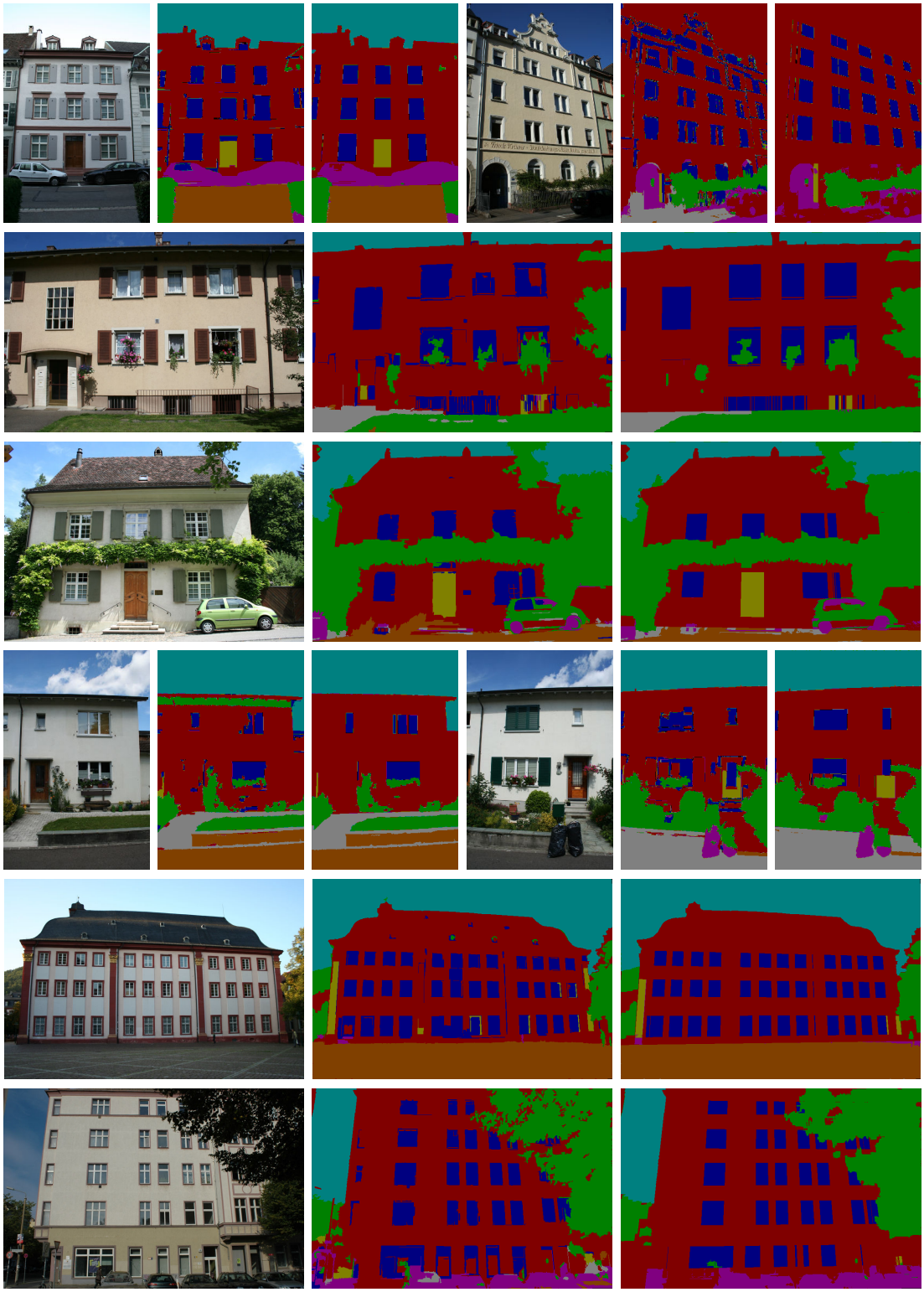


Figure 5.11: Example parsing results on the eTrims dataset in the order: original image, result of per-pixel classification, parsing result. Parsing was performed on rectified images which are presented here un-rectified.

Table 5.6: Performance on the eTrims dataset with RNN-based unaries. Starting from left: score using raw unaries, the three-layered method of Martinović et al. [33], the dynamic programming method of Cohen et al. [9] and our adjacency patterns (AP).

	eTrims			
	raw	[33]-L3	[9]	AP
building	88	87	91	92
car	69	69	70	70
door	25	19	18	20
pavement	34	34	33	33
road	56	56	57	56
sky	94	94	97	96
vegetation	89	88	90	91
window	71	79	71	70
pixel accur.	81.9	81.6	**83.8**	83.5

of the dimensions. One way to decrease the number of such errors is by means of pairwise potentials penalizing transitions between some classes. In this context, it would be useful to learn the optimal values of these penalties automatically, instead of determining them using grid search. One could also consider imposing constraints, or soft preference, on the size of elements. Doing that within the proposed framework would require an impractically large number of classes, equal to the expected size of the modeled object in pixels. Such approach would also impede generalization of the algorithm to unrectified images, where the size varies depending on the distance from the camera. Imposing constraints on relative sizes of facade elements seems more useful in that respect. However, this would require the use of higher order potentials and remains an open problem.

5.10 Conclusion

We have shown how complex, grid-structured patterns, possibly with irregular boundaries between regions corresponding to different semantic classes, can be encoded by specifying which pairs of classes can be assigned to pairs of vertically- and horizontally-adjacent pixels. We have argued that these patterns can be specified more conveniently in a hierarchical fashion and shown that the induced flattened set of rules can automatically be translated into the structure of a Markov random field.

Our method offers a unique combination of advantages that cannot be matched by other existing algorithms. The framework can accept user-defined priors and guarantee their satisfaction. The proposed prior formalism enables encoding winding boundaries between facade elements. Our formulation makes it possible to easily handle occlusions simultaneously segmenting the occluding objects and retrieving the structure of the occluded facade. Finally, the formulation lends itself to an efficient optimization scheme, resulting in highly accurate segmentations.

By testing on datasets containing complex facade patterns and occlusions, we demonstrated that our approach brings facade parsing with user-defined shape priors closer to practical applications. One possibility to extend the presented framework

is to learn the pairwise costs automatically. We also believe it is possible to learn a complete prior in form of an adjacency pattern from training data. In our opinion, the most exciting direction of research is extending the proposed framework to parsing non rectified images of complete urban scenes.

Publications

The work presented in this chapter has been published in

Koziński, M., Gadde, R., Zagoruyko, S., Obozinski, G., Marlet, R. (2015). A MRF Shape Prior for Facade Parsing with Occlusions. In *Proc. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Chapter 6

Conclusion and Future Work

The focus of this work is on leveraging efficient optimization algorithms to facade parsing with user-defined shape priors. Existing approaches to this problem are based on generative shape priors, called split grammars. They provide convenient means of encoding alignment, order of facade elements and their absolute sizes. While the framework of split grammars can be used to efficiently generate shapes, inverse modeling using such a prior is a difficult problem. Existing approaches resort to combinatorial algorithms, which require severe image subsampling, or randomly explore the space of grammar derivations, by generating a large number of segmentations and keeping the best one as the final solution. While the efficiency of this random exploration depends on the quality of bottom-up cues that drive the search towards optimum (appendix A), the random nature of the algorithm influences adversely the accuracy of resulting segmentations.

In this work, we formulate shape prior formalisms that have practical expressive power similar to that of split grammars, but that lend themselves to more principled optimization techniques, not requiring image subsampling or random exploration of the problem space. Even though the resulting problem of facade parsing is formulated as a binary linear program, or MAP-MRF inference, which are known to be NP-hard in general, experiment results show that, given the same data terms, the proposed algorithms attain better segmentations than competing methods.

One can think of the problem in terms of a tradeoff between the expressive power of a shape prior formalism and its practical aspects: capability of approximating optimal segmentations consistent with the prior and ease of encoding different priors. The proposed formalism combining graph grammars for modeling facade structure and MAP-MRF inference of optimal facade layout (chapter 3) enables encoding relative sizes of facade elements, but needs sampling facade structures from a graph grammar. Even though the formulation based on row and column labeling (chapter 4) does not encode the prior on element sizes, it attains better segmentations given the same data term, because it offers a more effective optimization scheme. Finally, the priors formulated using adjacency patterns (chapter 5) require the explicit specification of alignment of distant elements of the same class, but result in simpler formulation for finding the optimal segmentation. In consequence, this method attains better segmentations that the row and column label-based algorithm and enables formulating more flexible priors.

A shape prior can be treated as a regularization term defined on image segmentation. While some previous work [33] reported that imposing structure over segmentations comes at a cost of a decreased accuracy, in our experiments we have shown that applying parsing improves segmentation accuracies with respect to methods that are oblivious of the expected structure of the segmentation. We believe the capability of effectively optimizing the regularization term together with the data term is the key to fully benefit from the advantage of a shape prior.

6.1 Contribution

Our first contribution is a graphical model for facade segmentation, where a graph grammar formalism is used to represent the variation of model structure. We introduce an inference algorithm for the proposed setting, which combines sampling model structures from the grammar and a MAP-MRF inference for fixed structures. We have shown that limiting sampling to model structure improves the resulting segmentations when compared to split grammar parsing. However, due to the fact that sampling has not been eliminated completely, the approach is limited in practice with respect to the complexity of the grammar, because of the inefficiency of the random exploration. This constrains the level of architectural detail that can be modeled. That in turn adversely influences parsing accuracy and hinders application of the framework to datasets with a high level of structural variation.

The second major contribution presented in this work is a formulation for facade parsing with a shape prior format that enables representing a facade segmentation in terms of sets of classes assigned to image rows and columns. The priors encode simultaneous alignment of facade elements in two dimensions and the order of elements in a facade. The expressive power of the proposed prior formalism is different than that of split grammars, in that it does not offer a practical method for encoding constraints on sizes of facade elements, or patterns of facade elements expressed by context-free languages. However, sampling has been completely eliminated from inference, which has been formulated as a binary linear program. By solving the linear program we directly approximate the optimal segmentation, both in terms of the structure and the layout. This in turn results in an accuracy superior to that of the graph grammar method and better than for other existing algorithms, including ones with hard-coded priors. Unfortunately, the priors encoded using the proposed formalism, that enforces global 2D alignment between all facade elements of the same class, turn out to be highly restrictive in enforcing the global alignment of elements of the same class, which makes them impractical for modeling sets of shapes with a high degree of variability, possibly containing a large number of misaligned structures (like an unknown number of misaligned floors).

Finally, we propose a parsing framework where complex shape priors are encoded in terms of possible classes of adjacent pixels. Priors encoded using this approach feature unprecedented expressive power as they allow the modeling of elements with winding, non-linear boundaries. They also enable segmenting objects occluding the facade and extracting the structure of the occluded facade in the same time. This flexibility allowed us to use the framework on datasets featuring a level of structural variation that was prohibitively high for previous frameworks based on

user-defined priors. The task of determining the most likely segmentation within the proposed framework takes the form of MAP-MRF inference over a four-connected pixel grid. We propose to use the dual decomposition algorithm to approximate the optimum of the problem. We developed an efficient implementation of the algorithm, drawing on the fact that the matrices encoding allowed pairs of adjacent pixels are sparse, resulting in decreased running time with respect to the graph grammar and linear programming approaches. In terms of accuracy of resulting segmentations, the proposed algorithm outperforms our previous developments as well as other existing methods on a number of standard datasets of facade images.

With this work, we advance the applicability of facade parsing. When Teboul et al. presented the pioneering work on split grammar facade parsing [60, 59], their grammar simply encoded a perfect grid of windows, overlaid on horizontal stripes of sky, roof, wall and shop and included simple balconies (constrained to a single window, or running through whole facade width) and doors. The structural variation was limited to an unknown number of columns and floors, and an unknown type of balconies for each floor. The algorithms based on hard-coded shape priors, proposed later by Martinović et al. [33] and Cohen et al. [9] were more flexible in that they could produce much more complex segmentations. When combined with highly accurate data term they yielded considerably better segmentations than split grammars. However, they did so at the price of not satisfying all architectural constraints (such as the vertical alignment of windows). The proposed adjacency pattern-based prior formalism features enough expressive power to outperform these methods even when using the same data term. It combines the best of both words by guaranteeing conformance of the resulting segmentations to a user-defined shape prior and being almost as flexible as the algorithms by Cohen et al. and Martinović et al. in terms of modeled structure. Moreover, it addresses the problems of occlusions and complex-shaped facade elements, frequently appearing in realistic application scenarios.

6.2 Future work

Exploration of the space of possible facade structures in the graph grammar-based approach could benefit from bottom-up information driving the search for an optimal structure, based on facade parts detected in the image. This could speed up inference and enable using grammars with higher complexity. The existing bottom-up graph grammar parsing algorithms, like the one proposed by Rekers and Schurr [46], cannot handle missing or false positive nodes and edges, inevitable in computer vision applications. Therefore, a bottom-up graph grammar parser for facades remains an open problem.

In the proposed graph grammar parsing algorithm, a structure of a graphical model is perturbed in every iteration by regenerating a subtree of the derivation tree. Then MAP-MRF inference is performed for the new structure. A possible extension of the presented work is to explore the idea of speeding up the MAP-MRF inference by using information recorded during inference in previous iterations. This could be done by warm-starting the algorithm with state variables (messages, dual variables, depending on the inference algorithm used) corresponding to the final solution for

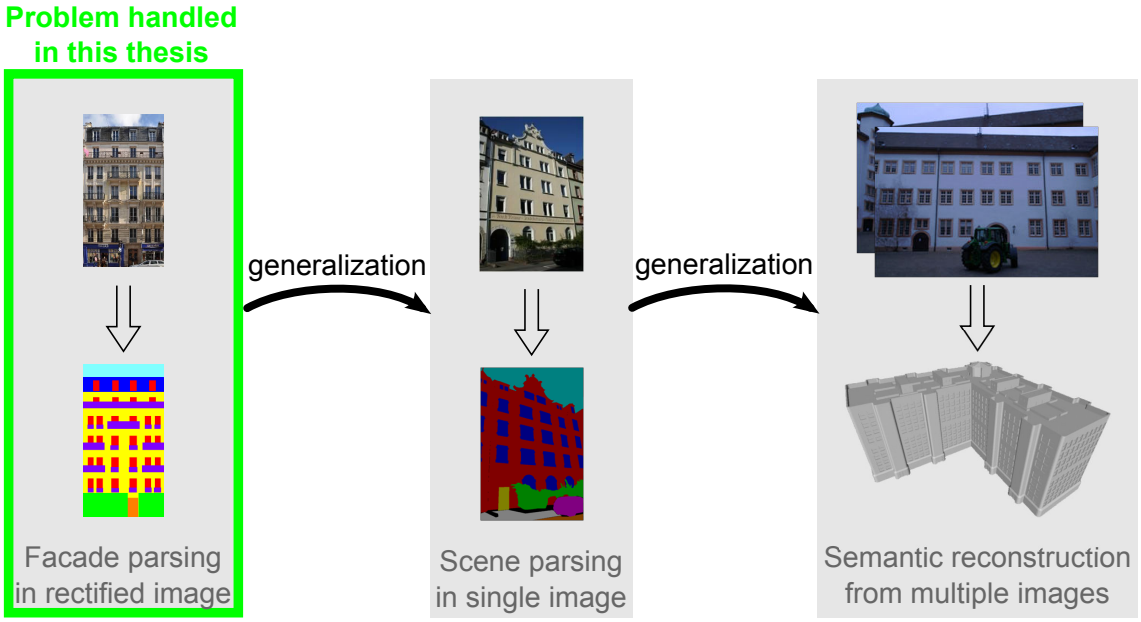


Figure 6.1: A possible plan for future work: a sequence of generalizations of the presented framework, leading from the task of parsing rectified images, to the task of reconstruction a semantized building model.

the model of previous structure. Such an approach could lead to a faster convergence in cases where the structure is modified only locally.

Since graph grammars can be difficult to specify, learning grammars from training data would increase the potential for practical application of graph grammar parsing. This task involves generating graphs from ground truth annotations and inferring a grammar given a large number of graphs. Again, one of the challenges lies in handling uncertainty in the graphs, as ground truth annotations can be imprecise.

For both the framework based on row and column labels, and the adjacency pattern-based framework, the most straightforward extension would be to learn the parameters of their energy functions from annotated examples. This can be done using the dual decomposition algorithm for max-margin learning proposed by Komodakis [25].

Another interesting direction of research would be to learn not only soft pairwise costs for pairs of classes of adjacent pixels, but the whole adjacency patterns. This approach requires extracting the pre-semantic classes and their possible adjacency configurations from ground truth annotations in terms of semantic classes. The task could be seen from the perspective of finding minimal automatons for image rows and columns, that would be capable of producing sequences of class labels that occur in the ground truth annotations. Again, the algorithm should be capable of handling imprecise annotations.

From the point of view of wide applicability, one of the most promising directions of research is to extend the framework of adjacency patterns to handling images of general urban scenes. One difficulty lies in modeling the projection on the image plane of complex three-dimensional building geometry, including street corners and multiple buildings. Encoding the three-dimensional alignment of scene elements as depicted in an image remains an open research question.

A step further would be to extend the framework to parsing multiple images of the same scene, resulting in a semantic reconstruction of a structurally correct 3D model. One possible approach to obtaining semantized, 'structured' models of buildings is to semantize meshes resulting from 3D reconstruction resulting from existing methods. Alternatively, one could try to create the models directly from images, by inferring the geometry and semantic structure of the building jointly. This approach could exploit the feedback between the semantic and geometric information. Reconstructing them jointly can increase the quality of the model in both aspects.

While this thesis does not treat the problem of 3D reconstruction, it exploits a number of approaches to obtaining semantized models from data, while imposing user-defined constraints on their structure. We believe these approaches can be extended to handle the more general problems of parsing general urban scenes and semantic 3D reconstruction. A possible sequence of generalization is visualized in figure 6.1.

Finally, we believe the adjacency pattern framework is applicable to other tasks, where models can be formulated in terms of constraints propagated along chains of data elements, like propagating alignment along rows and columns of pixels in case of facade parsing. One such task is segmenting point clouds of buildings, or whole urban scenes, where alignment in three dimensions plays a significant role.

Appendix A

Improved bottom-up cues for facade parsing

Here we present an attempt to address the limitations of the parser proposed by Teboul et al. [59], resulting from random exploration of the space of parse trees. Without altering the principle of operation of the parser itself, we propose bottom-up cues that effectively drive the exploration and speed up convergence. Instead of only using bottom-up cues in the form of local, per-pixel likelihoods, we propose to also use an object detector and a robust pattern search method. Our algorithm may thus exploit the repetition of specific instances of architectural elements within the facade. Additionally, to better guide parsing, we design prior distributions of possible split positions using the object detections and line segment cues. As a result, the parser not only better locates elements but also prunes the solution space much more selectively.

A.1 Enhanced per-pixel likelihoods

Instead of just using texture classification we propose a new, more robust and more accurate likelihood function, which combines the local, low-level (pixel- or patch-based) information with the results of object detection. In subsection A.1.1 we present the method we use for fusing detection results with texture classification-based potentials. In subsection A.1.2 we introduce an algorithm for discovering repeated objects, used for window detection.

In the following subsections we denote the set of indexes of image rows by I and the set of indexes of image columns by J. We define the set of image pixels by $\mathcal{I} = I \times J$.

A.1.1 Fusing detections and texture classification

The detections we use have a form of bounding boxes. In this chapter, we fuse the per-pixel potentials with detection results by projecting the detection bounding boxes to per-pixel weights (we present an alternative fusion scheme in section 3.4.1). Considering a class d, for each pixel (i, j) we get a weight $w_d(i, j)$, which we set to 1 if (i, j) belongs to the detected object, and to 0 otherwise.

Table A.1: Symbols used in this chapter

I	set of indexes of image rows; defined on page 107
J	set of indexes of image columns; defined on page 107
$\mathcal{I}$	set if indexes of image pixels; defined on page 107
P	a set if indexes of image pixels $P \subset \mathcal{I}$, defining a connected region in the image; defined on page 110
$\mathcal{C}$	set of classes assigned to pixels; defined on page 109
$\mathcal{D}$	set of classes of detected objects; defined on page 109
$C(d)$	a set of pixel classes that can overlap a detection of class d; defined on page 107
$C(D)$	a set of pixel classes that can overlap detections of all class in the set $D \subset \mathcal{D}$; defined on page 109
$w_d(i,j)$	a binary variable indicating the presence of detection of class $d \in \mathcal{D}$ whose bounding boxes overlap pixel (i,j); defined on page 107
$m(i,j,c)$	the merit function, expressing the likelihood of assigning class $c \in \mathcal{C}$ to pixel (i,j), based on texture classification; defined on page 109
$m_+(i,j,c)$	the modified merit function, expressing the likelihood of assigning class $c \in \mathcal{C}$ to pixel (i,j), based on texture classification and detection results; defined on page 109
L	the set of detected line segment indexes; defined on page 111
θ_l	the angle between detected line segment $l \in L$ and the horizontal coordinate axis; defined on page 111
$[a_l, b_l]$	the vertical line segment (with endpoints a_l and b_l) created by projecting the detected line segment $l \in L$ on the vertical axis; defined on page 111
$v(i)$	the distribution of horizontal split position at row $i \in I$; defined on page 111

In practice, the semantic classes assigned to pixels can be divided into two categories: 'things' and 'stuff' [19]. The first category represents objects that have a compact spatial extent and can be detected. The classes of the second kind occupy irregular image regions and are more naturally handled in terms of texture classification. For instance, we can detect windows or doors, but it is much harder to reliably detect walls or roofs. Moreover, although the segmentation algorithm assigns a single class to each image pixel, actual detectors can locate different objects that overlap in an image. For example, a bounding box representing a detected window can encompass both window-only areas and cast iron balconies in front of windows. Therefore, we differentiate between the semantic classes $\mathcal{C}$ of the grammar and the semantic classes $\mathcal{D}$ of the detectors, and define $C(d) \subset \mathcal{C}$ as the subset of grammar classes that can be overlapped by a detection of class $d \in \mathcal{D}$. We extend this definition to a set of detectable classes $D \subset \mathcal{D}$, defining the subset of grammar classes that can be overlapped by a detection of any class $d \in D$ by $C(D) = \bigcup_{d \in D} C(d)$.

We recall that the original parser [59] uses a per-pixel classification confidence function $m(i, j, c)$, that defines the likelihood that a pixel (i, j) represents a facade element of class c. In the remaining part of this section, we follow the naming convention of Teboul et al. [59], who call m a merit function. The parser looks for a derivation that minimizes the following total cost

$$E(\mathbf{c}) = \sum_{(i,j) \in \mathcal{I}} -\log\left(m(i, j, c_{ij})\right) , \tag{A.1}$$

where $\mathbf{c}$ is the vector of class labels for all image pixels, induced by the derivation.

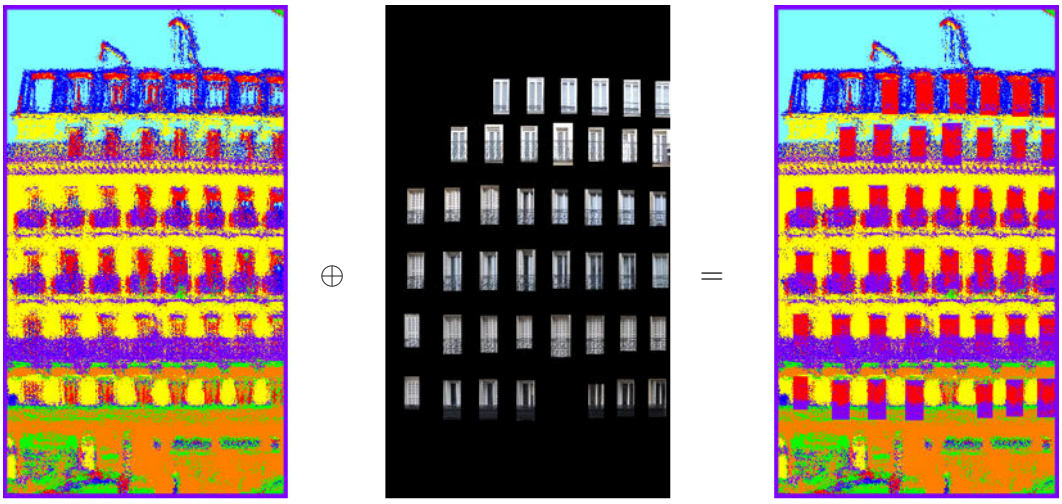
We construct an improved merit function, denoted m_+. It gives confidence to the high-level detectors over the low-level per-pixel classification. If a bounding box of detection of class d overlaps a given pixel, the merit for that pixel, for grammar classes that cannot overlap detector class d, is set to zero, and the merit for remaining classes is renormalized. More formally, let $D_{i,j} = \{d \in D \mid w_d(i, j) = 1\}$ be the set of classes detected at pixel (i, j). We define $m_+(i, j, c) = m(i, j, c)$ if $D_{i,j} = \emptyset$, i.e., the merit is unchanged for pixels which are not overlapped by any detection. Otherwise, if $D_{i,j} \neq \emptyset$, then $m_+(i, j, c)$ is defined as

$$m_+(i, j, c) = \begin{cases} \dfrac{m(i, j, c)}{\sum_{c' \in C(D_{i,j})} m(i, j, c')} & \text{if } c \in C(D_{i,j}) \\ 0 & \text{otherwise.} \end{cases} \tag{A.2}$$

In our experiments we trained a window detector that produces bounding boxes overlapping the whole window frame, even when there is a cast iron balcony in the foreground. We thus have $D = \{\text{whole-window}\}$ and $C(D) = \{\text{window}, \text{balcony}\}$. Figure A.1 illustrates the improved merit function. The most probable classes in the window areas are much less noisy.

A.1.2 Detection of repeated objects

Unlike in the task of general object detection, when detecting elements of a facade we can exploit the fact that in a given image we often expect a number of object



$m^*(x,y)$ $\quad$ $w_{\text{whole-window}}(x,y)$ $\quad$ $m^*_+(x,y)$

Figure A.1: Classification based on the local merit function (left) vs the higher-level merit function (right). We visualize the merit by color-coding the most probable class that can be assigned to each pixel according to $m^*(x,y,c) = \arg\max_c m(x,y,c)$ and $m^*_+ = \arg\max_c m_+(x,y,c)$. The result of window detection is presented in the middle.

instances that have similar appearance. This is true, for example, for windows in a given facade. Existing object detection methods, such as the cascade classifier [62], do not exploit this property. A hypothesis of these methods is that all detected objects are independent one from another.

We therefore propose to use a repeated pattern search scheme to extract a number of detections of consistent appearance in each image. We use a generic detector only to find few but very likely object occurrences, tuning the detector for precision more than recall. We then apply a robust pattern search procedure to find similar objects in the image. To improve recall though, this procedure is repeated recursively on the new detections (resulting from the pattern search) until there are no more high-precision detections.

Since we are working with rectified facade images, we assume that the detected objects are planar and that their instances are related by translation and anisotropic scaling. Indeed, windows on the same facade often have two or three different widths, depending on the size or use of the corresponding room. Although stretched horizontally, all windows on the facade however share similar appearance. Also for older structures, including Haussmannian buildings, bottom floors have higher ceilings and higher windows than top floors, to compensate for the weaker illumination. But here again the window appearance is only stretched, vertically. To make the pattern search more robust to deformations resulting from the parallax effect and imprecise detection of keypoints, we apply a pattern search procedure that looks for affine transformations of the model patch, but constrain the amount of shear and rotation present in the transformation. Specifically, we define the score of an affinity matrix $A = \begin{bmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \end{bmatrix}$ as $s(A) = \exp(-(a_2^2 + a_3^2))$ and only look for affinities with a score below a threshold $s_{\min}$.

The pattern search procedure is based on feature correspondences. Given the extent P of the pattern in image $\mathcal{I}$, correspondences to consider are only the pairs of

features (f_1, f_2) such that f_1 is in the model subimage P and f_2 is located in the rest of the image $\mathcal{I} \backslash P$. For robustness, we increase the area of the model P, by a constant factor, to include some context information. After patterns are localized, they are shrunk with the same factor to recover the estimated boundary of the actual model instances. We use RANSAC [45] to identify the transformations from the model patch to a new match. The sought transformation is characterized by a triplet of matches and to reduce the search space we pick a first match (f_1, f_2) and draw the two other matches (f_1', f_2') and (f_1'', f_2'') in such a way that f_2' and f_2'' lie within a neighborhood of f_2 in the image. More precisely, the distance between f_2 and f_2', and between f_2 and f_2'', has to be lower than the size of P multiplied by a fixed scale factor, constraining the scale variation of detected objects. The procedure is repeated until none of the model patches can be matched in the remaining part of the image with a predefined minimum number of inliers.

A.2 Enhanced distribution of split positions

We design a way to create more discriminative distributions for sampling horizontal and vertical split positions. In [59], these distributions are obtained by accumulating gradients in the image along the x and y axes. However, these marginal distributions are noisy because of the harmful accumulation of gradients not corresponding to objects of interest, but resulting from shadows, texture or small architectural details. We propose another approach, based on line segments detected in the image. These higher-level abstractions are better split indicators.

We first detect line segments L in the image. In our experiments we use the Line Segment Detector (LSD) [64]. Let $v(i)$ be the distribution of vertical split positions. We denote by $[a_l, b_l]$ the projection of a segment $l \in L$ on the vertical axis, and by θ_l its angle with respect to the horizontal coordinate axis. The value of the distribution at row i is computed as follows:

$$v(i) = \frac{1}{Z} \sum_{l \in L} \mathbb{1}_{i \in [a_l, b_l]} \exp(-\frac{\tan^2 \theta_l}{2\sigma^2}) \ , \tag{A.3}$$

where σ is a parameter of the distribution and Z is a normalization constant. The definition is symmetrical for horizontal splits. For our experiments, we set $\sigma = 0.06$, which roughly leads to a contribution of 1 for a perfectly axis-aligned segment, whereas a segment with a 5° angle contributes for $\frac{1}{3}$. To reduce computation time, line segments with an angle beyond a threshold (around 10° for the given σ value) can be discarded right after detection.

In the same manner, we build a normalized histogram of the contours of the detected objects. The two distributions are summed and the resulting histogram is normalized, yielding the final distribution of applicable split positions (see figure A.2). The major benefit of this approach is that the exploration of the solution space is significantly pruned and the splits are attracted to positions where detection boundaries and lines are present. Also, the parser is less likely to get stuck in a local minimum. In consequence, when the algorithm explores the procedural space, the standard deviation of the energy of solutions obtained in a number of consecutive iterations decreases over time faster than for the original algorithm (see figure A.3).

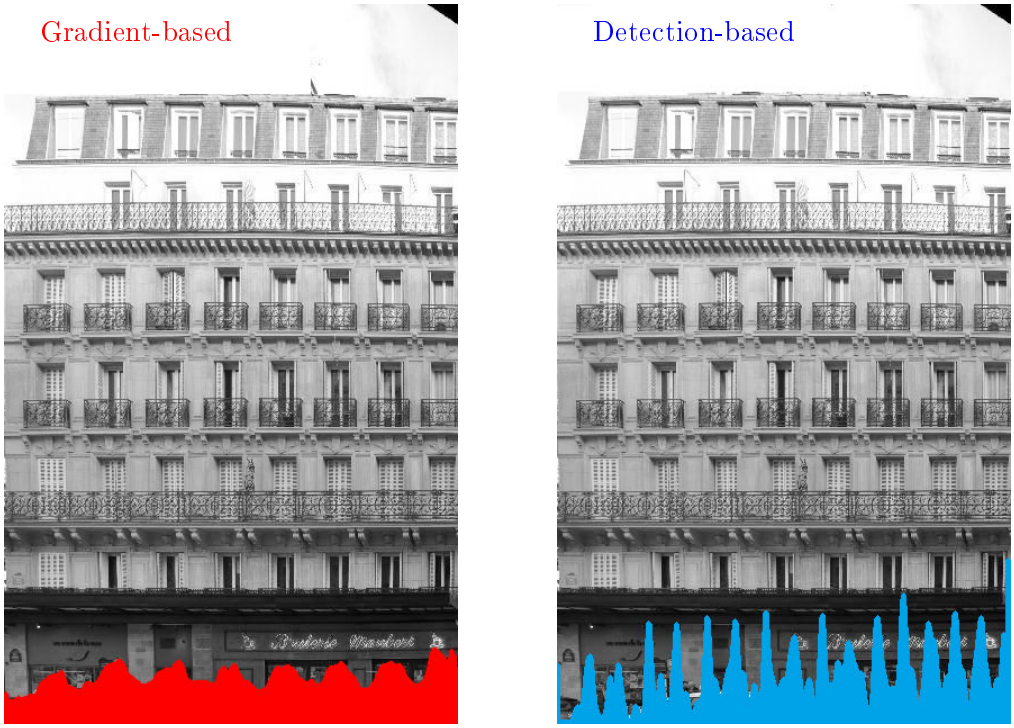


Figure A.2: The gradient-based vs the detection-based distribution of split positions.

A.3 Experimental validation

We perform the experimental validation on the ECP dataset [59], consisting of rectified images of Haussmannian facades, annotated with 7 semantic classes: sky, roof, wall, window, balcony, shop and door. When testing the performance of the repetitive pattern search scheme, we additionally use the eTrims dataset [27] containing images of buildings of many different architectural styles, with annotations for windows. We rectify the images manually. Before presenting parsing results, we first assess the quality of the window detector.

A.3.1 Window detection

First we verify the hypothesis that combining the proposed pattern search scheme with a precise detector yields better results than the detector alone. We obtain the initial detections with a cascade classifier (CC) [62]. We train the classifier to detect windows on 20 training images of the ECP CVPR 2010 dataset [60]. The classifier is parametrized by a threshold on the detection score τ, to balance precision and recall. We note by CC(τ) a detector resulting from using the cascade classifier with a threshold τ. For feature detection and matching, we use Harris-Affine [37], MSER [37] and DoG [32] detectors, and the SIFT [32] descriptor.

In tables A.2 and A.3 we present a comparison of the results of the cascade classifier for various detection thresholds (CC(τ)) to the results of the cascade classifier combined with our pattern search procedure (CC(τ)+PS). The methods are compared in terms of mean true positive rate ($\overline{\text{TPR}}$) and mean true negative rate ($\overline{\text{TNR}}$), measured on a per-pixel basis. In all datasets, applying the pattern search

Table A.2: Average performance of window detection on the ECP dataset. CC(τ) – the cascade classifier with threshold τ; PS – pattern search scheme. The classifier was trained on the 20 test images distinguished in the first version of the ECP dataset [60] and tested on the remaining images of the full ECP dataset [59].

Methods	$\overline{\textbf{TPR}}$ (%)	$\overline{\textbf{TNR}}$ (%)
CC ($\tau = 5$)	77	85.5
CC ($\tau = 10$)	70	81
CC ($\tau = 20$)	64	94
CC ($\tau = 30$)	56.5	95.5
CC ($\tau = 20$) + PS	76	94
CC ($\tau = 30$) + PS	71	95

Table A.3: Average performance of window detection on the rectified eTrims dataset. CC(τ) – the cascade classifier with threshold τ; PS – the pattern search scheme.

Methods	$\overline{\textbf{TPR}}$ (%)	$\overline{\textbf{TNR}}$ (%)
CC ($\tau = 5$)	46	96
CC ($\tau = 5$) + PS	60	93

procedure improves the $\overline{\text{TPR}}$ of CC by 12–15 points while maintaining the same $\overline{\text{TNR}}$. A slight decrease of $\overline{\text{TNR}}$ with respect to CC is only observed on the rectified eTRIMS dataset.

A.3.2 Facade parsing

To evaluate the impact of using the high-level cues on performance of facade parsing, we run the modified shape grammar parser on the test set of the ECP Benchmark 2011 dataset[1] (104 images). The window detector we used is CC($\tau = 20$)+PS, that experimentally performs best (see Table A.2). We compare this parser against the original one, presented in [59]. In each case we run the parsers once. The results

[1]This dataset must *not* be confused with *the ECP CVPR 2010 dataset* which consists of 10 test images only. Hence the numbers differ from what is reported in [59].

Table A.4: Parsing accuracy of the original parser of Teboul et al. [59], and of the same parser when using the proposed high-level cues (HLC). The rows of the table contain class-wise accuracies. The bottom row contains the total percentage of correctly classified pixels.

	[59]	[59]+HLC
roof	74	77
shop	82	82
balcony	50	60
sky	91	91
window	60	85
door	51	52
wall	81	76
pixel accur.	75.0	77.4

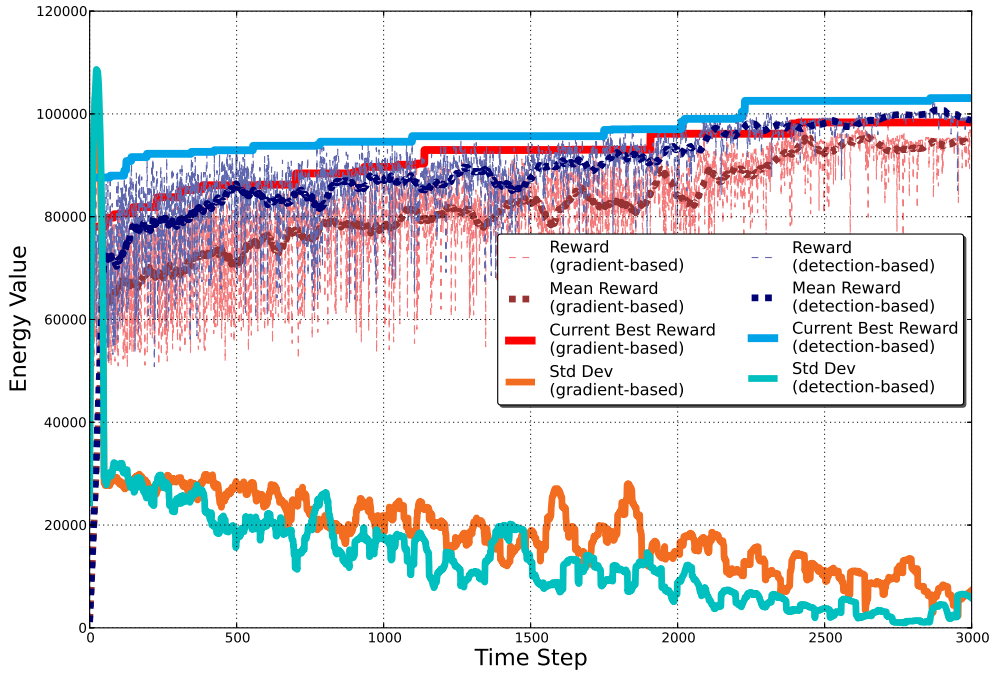


Figure A.3: The convergence of our algorithm in comparison to [59]. The plot shows the evolution of the reward, the current best reward and the standard deviation of the reward over time for a single run of the parser. The 'mean reward' is the plot of the reward function smoothed over time, to eliminate the high-frequency variation.

are evaluated with use of the ground truth annotations accompanying the dataset.

We present the results in table A.4. We observe an improvement of two percent in the total per-pixel accuracy, and an increase of the class-wise accuracies for most classes. In particular, the window detection improves from 60% to 85%. Our algorithm also shows better convergence properties than the original one. In figure A.3 we show that the proposed algorithm converges faster, attains better values of the reward function and is less prone to deviating from the optimal solution. A few actual results are illustrated in figure A.4.

A.4 Conclusion

We have presented a method to enhance the top-down facade parser of Teboul et al. [59]. We propose two modifications with respect to the original version. First, we propose to fuse results of an adaptive object detector with texture classification-based likelihoods to obtain an improved merit function, as opposed to relying on texture classification alone. Second, we use the boundaries of detection bounding boxes as well as line segments to better determine likely split positions. We have shown that these high-level cues improve parsing accuracy and speed up convergence of the parsing algorithm.

However, the observed improvement is marginal. Although the proposed cues guide the parser to better solutions, they do not address the inherent drawback of randomized exploration of the space of parse trees, consisting in yielding suboptimal, non-repeatable results. The susceptibility of the original parser to get stuck in local

optima is illustrated by the fact that when running experiments the authors re-run the algorithm five times and keep the best of the five results as the final segmentation [58]. The problem persists and is illustrated in figure A.3. The variance of energy yielded by the random exploration algorithm is comparable for the original method and for the proposed high-level bottom-up cues.

Contribution of the author

The presented work has been performed jointly with David Ok. The author of this manuscript contributed to the formulation of the detection-texture classification fusion scheme and the formulation of the enhanced distribution of split positions.

Publications

The work has been published in

Ok, D., Koziński, M., Marlet, R., Paragios, N. (2012). High-Level Bottom-Up Cues for Top-Down Parsing of Facade Images. In *Proc. 2nd Joint Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT).*

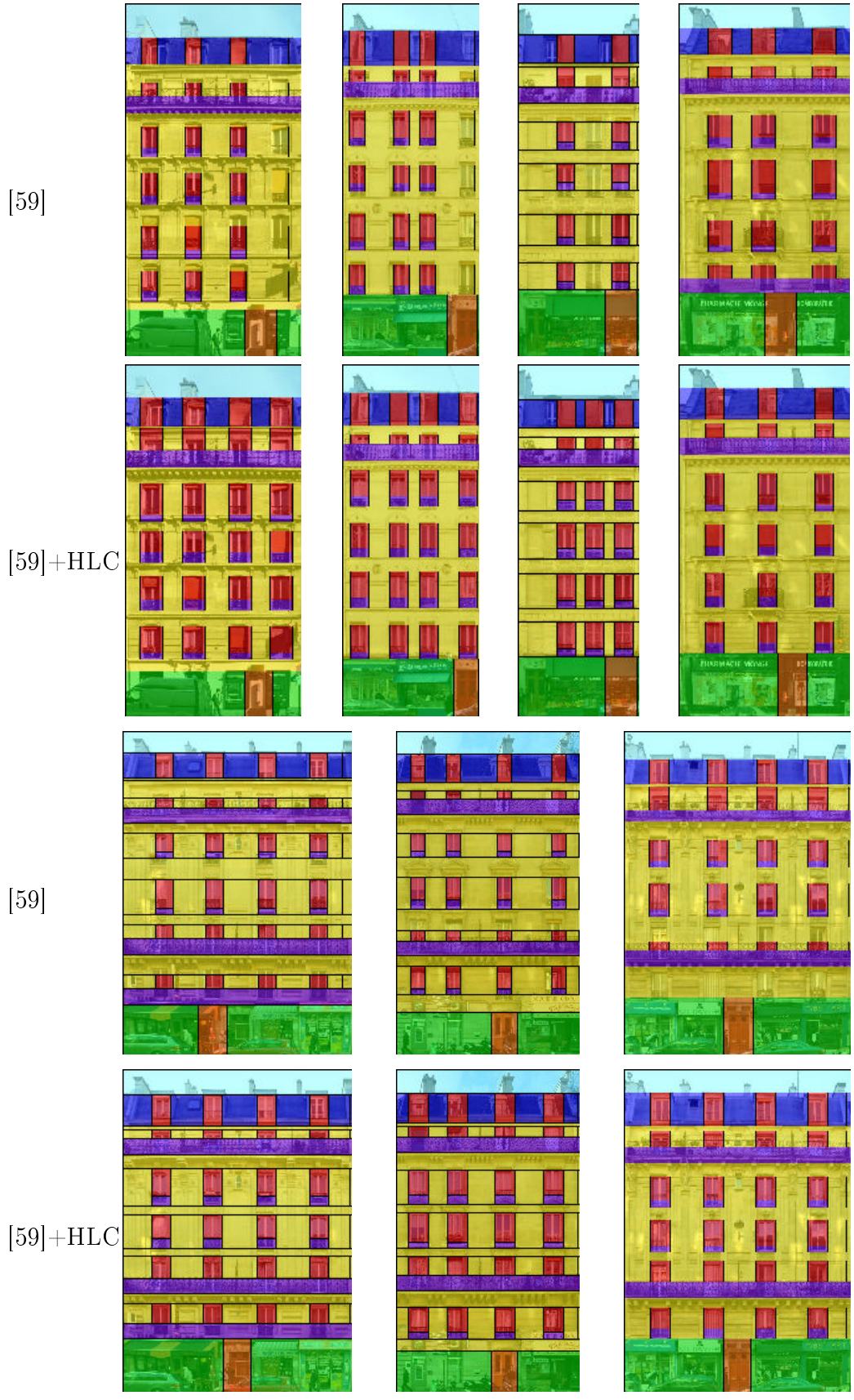


Figure A.4: Examples of images for which our parser outperforms the original algorithm. Odd rows: results of original parser [59]. Even rows: our modified algorithm, relying on high-level cues (HLC).

Appendix B

The Dual Decomposition algorithm

We briefly introduce the principle of the Dual Decomposition algorithm, that we use to solve problems posed in chapters 4 and 5. For a more formal, in depth treatment, we refer the reader to a comprehensive article by Komodakis, Paragios and Tziritas [26] and a tutorial by Sontag, Globerson and Jaakkola [54].

The dual decomposition algorithm is based on the idea of decomposing a difficult problem into a number of 'slave' subproblems that are easy to solve. We are given a problem of the form

$$\hat{\mathbf{x}} = \arg\min \sum_{m \in M} E_m(\mathbf{x}), \quad \text{s.t.} \quad \mathbf{x} \in C, \tag{B.1}$$

where $\mathbf{x}$ is a vector, C denotes the feasible set and M is a set of indeces m of components of the objective. We assume the objective is not easily optimized, but each of the component functions $E_m(\mathbf{x})$ can be efficiently minimized subject to $\mathbf{x} \in C$. In Dual Decomposition, (B.1) is solved by iteratively minimizing E_m separately for each m. We construct a number of copies of the vector $\mathbf{x}$, denoted $\mathbf{x}_m$, and couple the variables by means of a new constraint $\mathbf{x}_m = \mathbf{x}$:

$$(\hat{\mathbf{x}}_m)_{m \in M} = \arg\min \sum_{m \in M} E_m(\mathbf{x}_m) \quad , \tag{B.2}$$

subject to

$$\forall m, \quad \mathbf{x}_m \in C \quad , \tag{B.3}$$

$$\forall m, \quad \mathbf{x}_m = \mathbf{x} \quad . \tag{B.4}$$

Due to constraint (B.4) in the optimum of (B.2) we have $\hat{\mathbf{x}}_m = \hat{\mathbf{x}}$ for each m. Therefore, solving the modified problem can be seen as equivalent to solving the original one. To completely decouple objective components, we introduce vectors of Lagrangian multipliers $\boldsymbol{\lambda}_m$ and denote the vector created by concatenating $\boldsymbol{\lambda}_m$ for all $m \in M$ by $\boldsymbol{\lambda}$. We formulate a Lagrangian with respect to constraint (B.4)

$$\mathcal{L}(\boldsymbol{\lambda}) = \min_{(\mathbf{x}_m)_{m \in M}, \mathbf{x}} \sum_{m \in M} E_m(\mathbf{x}_m) + \sum_{m \in M} \boldsymbol{\lambda}_m^{\intercal}(\mathbf{x}_m - \mathbf{x}) \ , \quad \text{s.t.} \quad \forall m \in M, \ \mathbf{x}_m \in C \ . \tag{B.5}$$

For readability we rewrite (B.5) as

$$\mathcal{L}(\boldsymbol{\lambda}) = \min_{(\mathbf{x}_m)_{m\in M},\mathbf{x}} \sum_{m\in M} \Big(E_m(\mathbf{x}_m)+\boldsymbol{\lambda}_m^\intercal \mathbf{x}_m\Big) - \Big(\sum_{m\in M} \boldsymbol{\lambda}_m\Big)^\intercal \mathbf{x}\ , \quad \text{s.t.} \quad \forall m \in M,\ \mathbf{x}_m \in C\ . \tag{B.6}$$

We note that the problem is unbounded in $\mathbf{x}$ unless it holds

$$\sum_{m\in M} \boldsymbol{\lambda}_m = 0\ , \tag{B.7}$$

for which case the variable $\mathbf{x}$ is eliminated from the objective. We introduce a new notation for the modified Lagrangian:

$$\tilde{\mathcal{L}}(\boldsymbol{\lambda}) = \min_{(\mathbf{x}_m)_{m\in M}} \sum_{m\in M} \Big(E_m(\mathbf{x}_m) + \boldsymbol{\lambda}_m^\intercal \mathbf{x}_m\Big)\ , \quad \text{s.t.} \quad \forall m \in M, \quad \mathbf{x}_m \in C\ . \tag{B.8}$$

We emphasise $\tilde{\mathcal{L}}(\boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\lambda})$ when (B.7) is satisfied. Therefore the following problem is dual to (B.1)

$$\max_{\boldsymbol{\lambda}} \tilde{\mathcal{L}}(\boldsymbol{\lambda}) \quad , \tag{B.9}$$

subject to

$$\sum_{m\in M} \boldsymbol{\lambda}_m = 0 \quad . \tag{B.10}$$

The above dual objective is concave and a projected subgradient ascent algorithm can be used to solve it. In each iteration n of the algorithm, the dual variables $\boldsymbol{\lambda}_m$ are updated along the gradient direction $\nabla_{\boldsymbol{\lambda}_m}\tilde{\mathcal{L}}(\boldsymbol{\lambda})$ with a stepsize of predefined length α^n and reprojected to the set where (B.10) is satisfied, according to

$$\boldsymbol{\lambda}_m^n \leftarrow \Big[\boldsymbol{\lambda}_m^{n-1} + \alpha^n \nabla_{\boldsymbol{\lambda}_m}\tilde{\mathcal{L}}(\boldsymbol{\lambda}^{n-1})\Big]_{\sum_{m\in M}\boldsymbol{\lambda}_m=0}\ , \tag{B.11}$$

where the superscript n denotes the iteration index and $[\cdot]_C$ denotes a projection onto set C. The main advantage of this dual formulation, and the key to the Dual Decomposition algorithm, is that calculating the subgradient of $\tilde{\mathcal{L}}(\lambda)$ decomposes into independent minimizations for each m, called slave problems. In fact we can rewrite (B.8) as:

$$\tilde{\mathcal{L}}(\boldsymbol{\lambda}) = \sum_{m\in M} \tilde{\mathcal{L}}_m(\boldsymbol{\lambda}_m)\ , \tag{B.12}$$

where

$$\tilde{\mathcal{L}}_m(\boldsymbol{\lambda}_m) = \min_{\mathbf{x}_m} E_m(\mathbf{x}_m) + \boldsymbol{\lambda}_m^\intercal \mathbf{x}_m\ , \quad \text{s.t.} \quad \mathbf{x}_m \in C\ . \tag{B.13}$$

Each of the subgradients $\nabla_{\boldsymbol{\lambda}_m}\tilde{\mathcal{L}}(\boldsymbol{\lambda})$ for different m depends only on a single component of the objective, $\tilde{\mathcal{L}}_m(\boldsymbol{\lambda}_m)$, and can be calculated independently from the other ones:

$$\nabla_{\boldsymbol{\lambda}_m}\tilde{\mathcal{L}}(\boldsymbol{\lambda}) = \nabla_{\boldsymbol{\lambda}_m}\tilde{\mathcal{L}}_m(\boldsymbol{\lambda}_m) = \hat{\mathbf{x}}_m, \tag{B.14}$$

where

$$\hat{\mathbf{x}}_m = \arg\min_{\mathbf{x}_m} E_m(\mathbf{x}_m) + \boldsymbol{\lambda}_m^\intercal \mathbf{x}_m, \quad \text{s.t.} \quad \mathbf{x}_m \in C. \tag{B.15}$$

Algorithm 2 Dual Decomposition

$\forall m \in M \; \boldsymbol{\lambda}_m^0 \leftarrow 0$
$n \leftarrow 1$
while no convergence **do**
  $\forall m \in M \quad \hat{\mathbf{x}}_m^n \leftarrow \arg\min_{x_m} E_m(\mathbf{x}_m) + \boldsymbol{\lambda}_m^{n-1\intercal} \mathbf{x}_m$ ▷ Calc. subgr. (solve slaves)
  $\forall m \in M \quad \boldsymbol{\lambda}_m^n \leftarrow \boldsymbol{\lambda}_m^{n-1} + \alpha^n (\hat{\mathbf{x}}_m^n - \frac{1}{|M|} \sum_{m \in M} \hat{\mathbf{x}}_m^n)$ ▷ Step in subgr. dir. & repr.
  $n \leftarrow n + 1$
end while
$\hat{\mathbf{x}} \leftarrow \text{GETFINALX}((\hat{\mathbf{x}}_m^{n'})_{m \in M, 0 < n' < n}, (\boldsymbol{\lambda}_m^n)_{m \in M})$

The proof that $\hat{\mathbf{x}}_m$ is a subgradient of $\tilde{\mathcal{L}}_m(\boldsymbol{\lambda}_m)$ can be found in [26]. Summarizing, to calculate the subgradient, we need to solve problems of the form (B.14) independently for each m.

We detail the Dual Decomposition algorithm for solving (B.9) in algorithm 2. In each iteration the dual variables $\boldsymbol{\lambda}_m$ are updated along the direction of the subgradient. Reprojection of the dual variables into the feasible set, where constraint (B.10) is satisfied, is performed by subtracting their mean. In algorithm 2, the subgradient step and reprojection are combined into a single operation.

The algorithm is run for a number of iterations with decaying step size. Typically, a fixed sequence of step sizes is defined, which asymptotically goes to zero as n goes to infinity and with infinite sum at the limit, for example $\alpha^n = \frac{a}{\sqrt{n}}$ [26]. Convergence of the algorithm can be monitored by observing the changes of the dual energy. The values of most components of $\hat{\mathbf{x}}_m$ for different m eventually converge. Finally, a heuristic procedure, denoted in algorithm 2 as GETFINALX, can be used to decide on the components of $\mathbf{x}$ on which the vectors $\mathbf{x}_m$ for different $m \in M$ disagree [26]. Example heuristics include counting the number of times each variable was assigned each of the labels over a number of the last iterations and assigning the most frequent value to each of the variables, and fixing some of the variables while optimizing over the remaining ones in an iterative manner. The choice of a particular procedure is task-dependent.

The main design decision to be made when applying dual decomposition to an optimization problem is how to decompose the original objective function into slave objectives, i.e., how to transform it into the form (B.2) with a constraint of type (B.4). The main criterion is the ability to efficiently solve the resulting slave problems. Below we present a decomposition of the objective (4.17) into subproblems that can be solved by means of dynamic programming in time linear in the number of pixels.

Appendix C

Inference in the row- and column-labeling problem

In this appendix we derive the Dual-Decomposition-based algorithm used to solve the inference problem posed in chapter 4. The general Dual Decomposition approach has been introduced in appendix B.

C.1 Decomposition of the objective

In this section we present a decomposition of the objective (4.17) into the form (B.2). In order to represent minimization of the objective (4.17) subject to constraints (4.11) and (4.14–4.16) as a sum of tractable slaves, we need to decouple the variables y_{ik} corresponding to image rows from the variables x_{jl} corresponding to columns. One possible decomposition is into two problems, one assigning sets of classes to each image column, and other assigning classes to each image row. However, for a given column j, the number of possible configurations of vector $(x_{jl})_{l \in L}$ is combinatorial and grows exponentially with the number of vertically misaligned structures defined by shape prior. The same holds for feasible configurations of a set of row-classes assigned to image row. To avoid this exponential explosion of row and column labels, we propose to decompose the problem into a larger number of subproblems, one for each $l \in L \setminus T$ and one for each $k \in K \setminus T$. A subproblem indexed by $l \in L \setminus T$ assigns exactly one class from the set K_l to each image row and a subproblem indexed by $k \in K \setminus T$ assigns one class from the set L_k to each image column.

We introduce a separate set of variables for each of the subproblems, and we denote the variables with superscripts l and k, respectively. For each slave index $l \in L \setminus T$, for each $k \in K_l$ and $i \in I$, we introduce variables y^l_{ik}. Similarly, for each $k \in K \setminus \mathcal{T}$, we introduce variables x^k_{jl} defined for $l \in L_k$ and $j \in J$. We denote by $\mathbf{y}$ the vector created by stacking y_{ik}, by $\mathbf{y}^l$ the vector of y^l_{ik}, and by $\mathbf{x}$ and $\mathbf{x}^k$ the vectors obtained by stacking x_{jl} and x^k_{jl}, respectively. We denote the vectors of z_{ijt}, z^l_{ijt} and z^k_{ijt} by $\mathbf{z}$, $\mathbf{z}^l$ and $\mathbf{z}^k$, respectively. Since the costs c_{ijt} appear in all the slaves, they need to be distributed over the components of the objective in such a way that their sum is equal to the original cost. We distribute the costs evenly among $|K \setminus T| + |L \setminus T|$ slaves, resulting in cost coefficients of $\frac{c_{ijt}}{|K \setminus T| + |L \setminus T|}$ for each

slave. We introduce the components of the new objective

$$E^l(\mathbf{z}^l, \mathbf{y}^l) = \sum_{\substack{(i,j)\in\mathcal{I}\\ t\in T}} \frac{c_{ijt}}{|K\setminus T| + |L\setminus T|} z^l_{ijt} + \sum_{i=1}^{h-1} \sum_{k,k'\in Ch(l)} y^l_{ikk'} c_{kk'} \ , \tag{C.1a}$$

$$E^k(\mathbf{z}^k, \mathbf{x}^k) = \sum_{\substack{(i,j)\in\mathcal{I}\\ t\in T}} \frac{c_{ijt}}{|K\setminus T| + |L\setminus T|} z^k_{ijt} + \sum_{j=1}^{w-1} \sum_{l,l'\in Ch(k)} x^k_{jll'} c_{ll'} \ . \tag{C.1b}$$

We abuse the notation by omitting the vectors of pairwise variables $y^l_{ikk'}$ and $x^k_{jll'}$ in the list of arguments of E^l and E^k, because in our setting they will be completely determined by $\mathbf{y}^l$ and $\mathbf{x}^k$. We denote by $\mathbf{x}^K$ the concatenation of vectors $\mathbf{x}^k$ for $k \in K$, by $\mathbf{y}^L$ the concatenation of $\mathbf{y}^l$ for $l \in L$ and by $\mathbf{z}^L$, and $\mathbf{z}^K$ the concatenation of $\mathbf{z}^k$ and $\mathbf{z}^l$, respectively. The new objective becomes

$$E\big(\mathbf{z}^K, \mathbf{x}^K, \mathbf{z}^L, \mathbf{y}^L\big) = \sum_{k\in K\setminus T} E^k(\mathbf{z}^k, \mathbf{x}^k) + \sum_{l\in L\setminus T} E^l(\mathbf{z}^l, \mathbf{y}^l) \ . \tag{C.2}$$

Our optimization problem now consists in minimizing the objective

$$\min_{\mathbf{z}^K, \mathbf{x}^K, \mathbf{z}^L, \mathbf{y}^L} E\big(\mathbf{z}^K, \mathbf{x}^K, \mathbf{z}^L, \mathbf{y}^L\big) \tag{C.3}$$

subject to the non-negativity constraints on the relaxed variables:

$$\forall l \in L\setminus T \quad \mathbf{z}^l \geq 0, \quad \mathbf{y}^l \geq 0 \ , \tag{C.4a}$$

$$\forall k \in K\setminus T \quad \mathbf{z}^k \geq 0, \quad \mathbf{x}^k \geq 0 \ , \tag{C.4b}$$

the coupling constraints:

$$\forall k \in K\setminus T \quad \mathbf{z}^k = \mathbf{z} \ , \tag{C.5a}$$

$$\forall l \in L\setminus T \quad \mathbf{z}^l = \mathbf{z} \ , \tag{C.5b}$$

$$\forall l \in L\setminus T \quad \mathbf{y}^l = \mathbf{y} \ , \tag{C.5c}$$

$$\forall k \in K\setminus T \quad \mathbf{x}^k = \mathbf{x} \ , \tag{C.5d}$$

the constraints (4.11), enforcing the structure of the class hierarchy, on the new variables:

$$\forall i \in I,\ \forall l \in L\setminus T \quad \sum_{k\in K_l} y^l_{ik} = 1 \ , \tag{C.6a}$$

$$\forall j \in J,\ \forall k \in K\setminus T \quad \sum_{l\in L_k} x^k_{jl} = 1 \ , \tag{C.6b}$$

the constraints (4.14) and (4.15) on the newly introduced variables:

$$\forall (i,j) \in \mathcal{I},\ \forall l \in L\setminus T \quad \sum_{t\in T} z^l_{ijt} = 1 \ , \tag{C.7a}$$

$$\forall (i,j) \in \mathcal{I},\ \forall k \in K \setminus T \quad \sum_{t \in T} z^k_{ijt} = 1 \ , \tag{C.7b}$$

$$\forall (i,j) \in \mathcal{I},\ \forall l \in L \setminus T,\ \forall k \in K_l \quad \sum_{t \in Desc(k)} z^l_{ijt} \leq y^l_{ik} \ , \tag{C.8a}$$

$$\forall (i,j) \in \mathcal{I},\ \forall k \in K \setminus T,\ \forall l \in L_k \quad \sum_{t \in Desc(l)} z^k_{ijt} \leq x^k_{jl} \ . \tag{C.8b}$$

and the constraints on the pairwise variables:

$$\forall i \in I \setminus \{h\}, \forall l \in L \setminus T,\ \ \forall k \in K_l \quad \sum_{k' \in K_l} y^l_{ikk'} = y^l_{ik} \ , \tag{C.9a}$$

$$\forall i \in I \setminus \{h\}, \forall l \in L \setminus T,\ \forall k' \in K_l \quad \sum_{k \in K_l} y^l_{ikk'} \ = y^l_{i+1\,k'} \ , \tag{C.9b}$$

$$\forall j \in J \setminus \{w\}, \forall k \in K \setminus T,\ \forall l \in L_k \quad \sum_{l' \in L_k} x^k_{jll'} \ = x^k_{jl} \ , \tag{C.9c}$$

$$\forall j \in J \setminus \{w\}, \forall k \in K \setminus T, \forall l' \in L_k \quad \sum_{l \in L_k} x^k_{jll'} \ = x^k_{j+1\,l'} \ . \tag{C.9d}$$

We note that the domains of objective components $E^l(\mathbf{z}^l, \mathbf{y}^l)$ and $E^k(\mathbf{z}^k, \mathbf{x}^k)$ are coupled only by constraints (C.5).

C.2 Formulation of the dual problem

In this section, we present a dual formulation of the problem defined by equations (C.3–C.9). The transformations presented here are a straightforward application of the Dual Decomposition framework presented in appendix B.

We denote the vector created by stacking all $(\mathbf{z}^l, \mathbf{z}^k, \mathbf{z})$ by $\breve{\mathbf{z}}$. Similarly by $\breve{\mathbf{y}}$ we denote all $y^l_{ik}, y^l_{ikk'}, y_{ik}$ and the vector of all $x^k_{jl}, x^k_{jll'}, x_{jl}$ is denoted by $\breve{\mathbf{x}}$. We introduce the dual variables λ^l_{ijt} and λ^k_{ijt}, corresponding to constraints (C.5a) and (C.5b), and denote the vector resulting from stacking them together by $\boldsymbol{\lambda}$, and the vectors obtained by stacking variables with the same superscripts by $\boldsymbol{\lambda}^l$ and $\boldsymbol{\lambda}^k$. We also introduce dual variables γ^l_{ik} for $l \in L \setminus T$ and $k \in K_l$, and γ^k_{jl}, defined for $k \in K \setminus T$ and $l \in L_l$. The variables are Lagrange multipliers corresponding to constraints (C.5c) and (C.5d). We denote the vector obtained by stacking them together by $\boldsymbol{\gamma}$, and the vectors resulting from stacking variables with the same superscripts by $\boldsymbol{\gamma}^l$ and $\boldsymbol{\gamma}^k$.

The application of the dual decomposition method to objective (C.3) consists in formulating its Lagrangian with respect to the coupling constraints (C.5):

$$\begin{aligned} L_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \min_{\breve{\mathbf{z}}, \breve{\mathbf{y}}, \breve{\mathbf{x}}} & \sum_{k \in K \setminus T} E^k(\mathbf{z}^k, \mathbf{x}^k) + \sum_{l \in L \setminus T} E^l(\mathbf{z}^l, \mathbf{y}^l) \\ & + \sum_{l \in L \setminus T} \langle \boldsymbol{\lambda}^l, (\mathbf{z}^l - \mathbf{z}) \rangle + \sum_{k \in K \setminus T} \langle \boldsymbol{\lambda}^k, (\mathbf{z}^k - \mathbf{z}) \rangle \\ & + \sum_{l \in L \setminus T} \sum_{\substack{i \in I \\ k \in K_l}} \gamma^l_{ik} (y^l_{ik} - y_{ik}) + \sum_{k \in K \setminus T} \sum_{\substack{j \in J \\ l \in L_k}} \gamma^k_{jl} (x^k_{jl} - x_{jl}) \ , \end{aligned} \tag{C.10}$$

subject to constraints (C.4) and (C.6–C.9), where by $\langle \cdot, \cdot \rangle$ we denote the inner product. As in the general case presented in appendix B, the variable vector $\mathbf{z}$ can be eliminated by ensuring that

$$\sum_{l \in L \setminus T} \boldsymbol{\lambda}^l + \sum_{k \in K \setminus T} \boldsymbol{\lambda}^k = 0 \,, \tag{C.11a}$$

and the variables y_{ik} and x_{jl} are eliminated by setting

$$\forall i \in I, \forall k \in K, \sum_{l \in L \setminus T \text{ s.t. } k \in K_l} \gamma^l_{ik} = 0 \quad \text{and} \quad \forall j \in J, \forall l \in L, \sum_{k \in K \setminus T \text{ s.t. } l \in L_k} \gamma^k_{jl} = 0 \quad , \tag{C.11b}$$

since if (C.11) does not hold, the minimum in $\mathbf{z}$, $\mathbf{y}$ and $\mathbf{x}$ is infinite. By eliminating $\mathbf{z}$, $\mathbf{y}$ and $\mathbf{x}$ from (C.10) we can create the modified Lagrangian which decomposes into independent minimizations for each $k \in K \setminus T$ and $l \in L \setminus T$:

$$\tilde{L}_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \sum_{k \in K \setminus T} \tilde{L}^k_D(\boldsymbol{\lambda}^k, \boldsymbol{\gamma}^k) + \sum_{l \in L \setminus T} \tilde{L}^l_D(\boldsymbol{\lambda}^l, \boldsymbol{\gamma}^l) \,, \tag{C.12}$$

where

$$\tilde{L}^k_D(\boldsymbol{\lambda}^k, \boldsymbol{\gamma}^k) = \min_{\mathbf{z}^k, \mathbf{x}^k, \tilde{\mathbf{x}}^k} E^k(\mathbf{z}^k, \mathbf{x}^k) + \langle \boldsymbol{\lambda}^k, \mathbf{z}^k \rangle + \langle \boldsymbol{\gamma}^k, \mathbf{x}^k \rangle \,, \tag{C.13}$$

$$\tilde{L}^l_D(\boldsymbol{\lambda}^l, \boldsymbol{\gamma}^l) = \min_{\mathbf{z}^l, \mathbf{y}^l, \tilde{\mathbf{y}}^l} E^l(\mathbf{z}^l, \mathbf{y}^l) + \langle \boldsymbol{\lambda}^l, \mathbf{z}^l \rangle + \langle \boldsymbol{\gamma}^l, \mathbf{y}^l \rangle \,, \tag{C.14}$$

and both minimizations are subject to constraints (C.4) and (C.6–C.9).

The final form of the dual problem is

$$\max_{\boldsymbol{\lambda}} \; \tilde{L}_D(\boldsymbol{\lambda}) \,, \tag{C.15}$$

subject to constraint (C.11). We solve the problem by means of a projected subgradient-ascent procedure presented in algorithm 2. In each iteration, we update the dual variables $\boldsymbol{\lambda}^l$, $\boldsymbol{\lambda}^k$, $\boldsymbol{\gamma}^l$ and $\boldsymbol{\gamma}^k$ by making a step in the direction of the subgradient and reprojecting them into the feasible set, where the constraint (C.11) is satisfied. In the next section, we derive the update equations.

C.3 The optimization algorithm

We solve (C.15) by a projected subgradient method. By an argument identical to the one presented in appendix B, it can be shown that a subdifferential of (C.15) with respect to the dual variables contains the following vectors:

$$\nabla_{\boldsymbol{\lambda}^k} \tilde{L}_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{\mathbf{z}}^k, \qquad \nabla_{\boldsymbol{\lambda}^l} \tilde{L}_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{\mathbf{z}}^l, \tag{C.16a}$$

$$\nabla_{\boldsymbol{\gamma}^k} \tilde{L}_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{\mathbf{x}}^k, \qquad \nabla_{\boldsymbol{\gamma}^l} \tilde{L}_D(\boldsymbol{\lambda}, \boldsymbol{\gamma}) = \hat{\mathbf{y}}^l, \tag{C.16b}$$

where

$$(\hat{\mathbf{z}}^k, \hat{\mathbf{x}}^k) = \arg\min_{\mathbf{z}^k, \mathbf{x}^k} E^k(\mathbf{z}^k, \mathbf{x}^k) + \langle \boldsymbol{\lambda}^k, \mathbf{z}^k \rangle + \langle \boldsymbol{\gamma}^k, \mathbf{x}^k \rangle \quad , \tag{C.17}$$

subject to (C.4) and (C.6–C.9), and

$$(\hat{\mathbf{z}}^l, \hat{\mathbf{y}}^l) = \arg\min_{\mathbf{z}^l, \mathbf{y}^l} E^l(\mathbf{z}^l, \mathbf{y}^l) + \langle \boldsymbol{\lambda}^l, \mathbf{z}^l \rangle + \langle \boldsymbol{\gamma}^l, \mathbf{y}^l \rangle \quad , \tag{C.18}$$

subject to subject to (C.4) and (C.6–C.9).

The algorithm 2 adapted to the dual problem (C.15) takes the form presented in algorithm 3. We denote the values of variables at iteration n with a superscript. We denote the vectors obtained by stacking $\hat{\mathbf{z}}^k$ and $\hat{\mathbf{z}}^l$ for all $k \in K \setminus T$ and all $l \in L \setminus T$ by $\hat{\mathbf{z}}$. We denote the vector created by stacking $\hat{\mathbf{x}}^k$ for all $k \in K \setminus T$ by $\hat{\mathbf{x}}$ and the vector created by stacking $\hat{\mathbf{y}}^l$ for all $l \in K \setminus T$ by $\hat{\mathbf{y}}$. The dual variables are updated by a step along the subgradient direction and reprojected to the feasible set, where constraints (C.11) are satisfied. The reprojection consists in subtracting the average from the corresponding variables. We denote the average of all slave solutions at iteration n by

$$\bar{z}^n_{ijt} = \frac{1}{|L \setminus T| + |K \setminus T|} \left(\sum_{l \in L \setminus T} \hat{z}^{l,n}_{ijt} + \sum_{k \in K \setminus T} \hat{z}^{k,n}_{ijt} \right) \quad , \tag{C.19}$$

$$\bar{y}^n_{ik} = \frac{1}{|L(k)|} \sum_{l' \in L(k)} \hat{y}^{l',n}_{ik} \, , \tag{C.20}$$

$$\bar{x}^n_{jl} = \frac{1}{|K(l)|} \sum_{k' \in K(l)} \hat{x}^{k',n}_{jl} \, , \tag{C.21}$$

where $L(k) = \{l \in L \setminus T | k \in K_l\}$ and $K(l) = \{k \in K \setminus T | l \in L_k\}$.

Algorithm 3 Dual Decomposition applied to the problem

$\forall l \in L \setminus T, \quad \boldsymbol{\lambda}^0_l \leftarrow 0, \; \boldsymbol{\gamma}^0_l \leftarrow 0$
$\forall k \in K \setminus T, \; \boldsymbol{\lambda}^0_k \leftarrow 0, \; \boldsymbol{\gamma}^0_k \leftarrow 0$
$n \leftarrow 1$
while no convergence **do**
  $\forall k \in K \setminus T \quad (\hat{\mathbf{z}}^{k,n}, \hat{\mathbf{x}}^{k,n}) \leftarrow \arg\min_{\mathbf{z}^k, \mathbf{x}^k} E^k(\mathbf{z}^k, \mathbf{x}^k) + \langle \boldsymbol{\lambda}^{k,n-1}, \mathbf{z}^k \rangle + \langle \boldsymbol{\gamma}^{k,n-1}, \mathbf{x}^k \rangle$
  $\forall l \in L \setminus T \quad (\hat{\mathbf{z}}^{l,n}, \hat{\mathbf{y}}^{l,n}) \leftarrow \arg\min_{\mathbf{z}^l, \mathbf{y}^l} E^l(\mathbf{z}^l, \mathbf{y}^l) + \langle \boldsymbol{\lambda}^{l,n-1}, \mathbf{z}^l \rangle + \langle \boldsymbol{\gamma}^{l,n-1}, \mathbf{y}^l \rangle$
  $\forall k \in K \setminus T, \; \forall t \in T, \forall (i,j) \in \mathcal{I} \quad \lambda^{k,n}_{ijt} \leftarrow \lambda^{k,n-1}_{ijt} + \alpha^n \left(\hat{z}^{k,n}_{ijt} - \bar{z}^n_{ijt} \right)$
  $\forall l \in L \setminus T, \; \forall t \in T, \forall (i,j) \in \mathcal{I} \quad \lambda^{l,n}_{ijt} \leftarrow \lambda^{l,n-1}_{ijt} + \alpha^n \left(\hat{z}^{l,n}_{ijt} - \bar{z}^n_{ijt} \right)$
  $\forall k \in K \setminus T, \; \forall l \in L_k, \forall j \in J \quad \gamma^{k,n}_{jl} \leftarrow \gamma^{k,n-1}_{jl} + \alpha^n \left(\hat{x}^{k,n}_{jl} - \bar{x}^n_{jl} \right)$
  $\forall l \in L \setminus T, \; \forall k \in K_l, \forall i \in I \quad \gamma^{l,n}_{ik} \leftarrow \gamma^{l,n-1}_{ik} + \alpha^n \left(\hat{y}^{l,n}_{ik} - \bar{y}^n_{ik} \right)$
  $n \leftarrow n + 1$
end while
$\hat{\mathbf{z}}, \hat{\mathbf{y}}, \hat{\mathbf{x}} \leftarrow$ GetPrimSol$((\hat{\mathbf{z}}^{n'})_{1 \le n' \le n}, (\hat{\mathbf{y}}^{n'})_{1 \le n' \le n}, (\hat{\mathbf{x}}^{n'})_{1 \le n' \le n}, (\lambda^{n'})_{1 \le n' \le n})$

We run the algorithm with a predefined sequence of decaying step size. In algorithm 3, the step size in iteration n is denoted α^n. At the end, copies of most variables take the same value across different slaves, and there is a small number of

variables, on which the slaves disagree. As explained before, the algorithm works on the dual problem. To extract the solution to the primal problem, we use a heuristic based on selecting for each variable the value most frequently assigned by the slaves [26]. However, when applied directly, this method does not guarantee satisfaction of constraint (4.15) of the original problem. We therefore apply a greedy optimization procedure, denoted GetPrimSol in algorithm 3, which produces two candidate labelings and selects the better of the two as the final solution. The first labeling is obtained by first setting the row labels according to the heuristics, and then, for fixed row labels, greedily setting the column labels to minimize the energy of the segmentation. The second one is generated by assigning the column labels first, and then selecting the row labels.

C.4 Structure of a slave subproblem

Below we present the structure of a slave subproblem (C.18) for some l. The slaves for k are created symmetrically. The copies of the variables are denoted with superscripts l. For notational convenience we introduce a new symbol for the per-pixel costs for the slaves $c^l_{ijt} = \frac{c_{ijt}}{(|L\setminus T|+|K\setminus T|)}$. Their sum over all the slaves is equal to the original cost coefficients c_{ijt}. The resulting objective is

$$\min_{\mathbf{z}^l,\mathbf{y}^l} E^l(\mathbf{z}^l,\mathbf{y}^l) = \sum_{\substack{(i,j)\in\mathcal{I}\\ t\in T}} (c^l_{ijt}+\lambda^l_{ijt})z^l_{ijt} + \sum_{\substack{i\in I\\ k\in K_l}} \gamma^l_{ik} y^l_{ik} + \sum_{\substack{i\in I\setminus\{h\}\\ k,k'\in Ch(l)}} c_{kk'} y^l_{ikk'} \quad , \tag{C.22}$$

subject to

$$\forall (i,j)\in\mathcal{I},\ \forall t\in T,\ z^l_{ijt}\geq 0\ , \qquad \forall (i,j)\in\mathcal{I},\ \sum_{t\in T} z^l_{ijt} = 1\ , \tag{C.23a}$$

$$\forall i\in I,\ \forall k\in K_l,\ y^l_{ik}\geq 0\ , \qquad \forall i\in I,\ \sum_{k\in K_l} y^l_{ik} = 1\ , \tag{C.23b}$$

$$\forall (i,j)\in\mathcal{I},\quad \forall k\in K_l,\quad \sum_{t\in Desc(k)} z^l_{ijt} \leq y^l_{ik}\ , \tag{C.23c}$$

$$\forall i\in I\setminus\{h\},\quad \forall k\in K_l,\quad \sum_{k'\in K_l} y^l_{ikk'} = y^l_{ik}\ , \tag{C.23d}$$

$$\forall i\in I\setminus\{h\},\quad \forall k'\in K_l,\quad \sum_{k\in K_l} y^l_{ikk'} = y^l_{i+1\,k'}\ . \tag{C.23e}$$

The feasible set of the slave problem, defined above in (C.23), is just a reformulation of constraints (C.4) and (C.6–C.9) which have variables with superscript l in their domain. We rewrote the constraints here for future reference.

C.5 Integrality of the Slave Subproblem

In this section, we show that all vertices of the feasible set defined by constraints (C.23) are integral. We notice that constraints (C.23b), (C.23d) and (C.23e) form

a feasible set of relaxation of a binary linear program for finding the most probable configuration of a Markov Chain. See, for example, the article by Werner [68] for a detailed treatment of this topic. Since relaxations of tree-structured graphical models are tight, it is enough to show that the objective remains linear in y^l_{ik} after optimizing over z^l_{ijt}.

To eliminate z^l_{ijt} from the slave objective (C.22), we rewrite it as

$$\min_{y^l_{ik}, y^l_{ikk'}} \sum_{(i,j)\in\mathcal{I}} g_{ij}(\mathbf{y}^l_i) + \sum_{\substack{i\in I\\ k\in K_l}} \lambda^l_{ik} y^l_{ik} + \sum_{\substack{i\in I\setminus\{h\}\\ k,k'\in Ch(l)}} c_{kk'} y^l_{ikk'} \quad , \tag{C.24}$$

subject to constraints (C.23b), (C.23d) and (C.23e). We recall that by $\mathbf{y}^l_i$ we denote a vector $(y^l_{ik})_{k\in K_l}$ for a given i and for all $k \in K_l$. We argue that $g_{ij}(\cdot)$ is a linear function.

The function $g_{ij}(\cdot)$ can be formulated as

$$g_{ij}(\mathbf{y}^l_i) = \min_{(z^l_{ijt})_{t\in T}} \sum_{t\in T} (c^l_{ijt} + \lambda^l_{ijt}) z^l_{ijt} \tag{C.25}$$

subject to

$$\forall t \in T,\ \ z^l_{ijt} \geq 0 \ , \tag{C.26}$$

$$\sum_{t\in T} z^l_{ijt} = 1 \ , \tag{C.27}$$

$$\forall k \in K_l, \quad \sum_{t\in Desc(k)} z^l_{ijt} \leq y^l_{ik} \ , \tag{C.28}$$

where the constraints (C.26–C.28) are rewritten from the original problem (C.23a) and (C.23c). To demonstrate certain properties of the feasible set, we reformulate the constraints. We rewrite (C.28) as an equality constraint, introducing auxiliary variables δ_{ijk}:

$$\forall k \in K_l, \quad \delta_{ijk} \geq 0 \ , \tag{C.29}$$

$$\forall k \in K_l, \quad \sum_{t\in Desc(k)} z^l_{ijt} + \delta_{ijk} = y^l_{ik} \ . \tag{C.30}$$

We sum both sides of (C.30) for all $k \in K_l$ and use the fact that, according to (C.23b), y^l_{ik} sum to one, to get

$$\sum_{k\in K_l} \sum_{t\in Desc(k)} z^l_{ijt} + \sum_{k\in K_l} \delta_{ijk} = 1 \tag{C.31}$$

We note that $Desc(k)$ for $k \in K_l$ are disjoint sets (see equation (4.10) for the definition of K_l). We denote the complement in T of their union as $T^*_l = T \setminus \bigcup_{k\in K_l} Desc(k)$. By moving the first component of (C.31) to the right-hand side, and applying (C.27) we get

$$\sum_{k\in K_l} \delta_{ijk} = \sum_{t\in T^*_l} z^l_{ijt} \ . \tag{C.32}$$

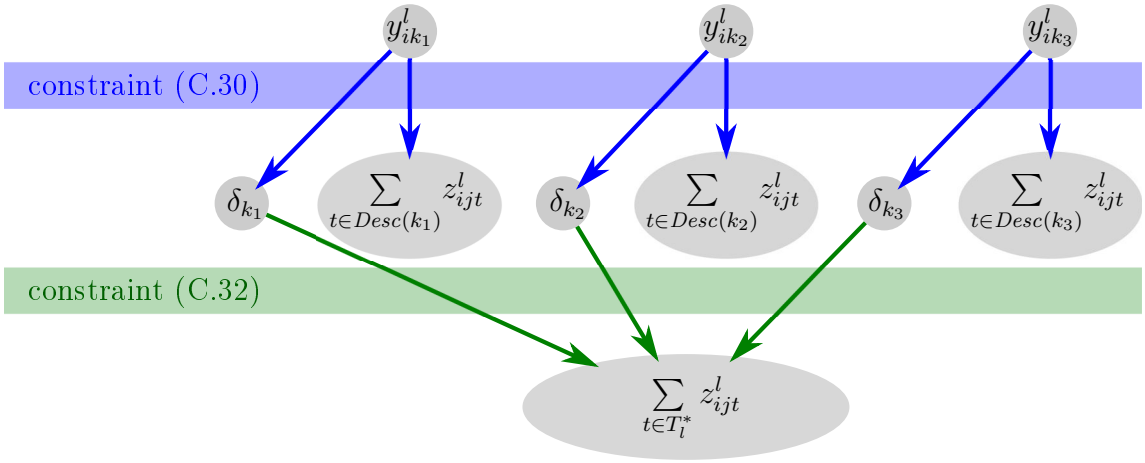


Figure C.1: Visualization of a possible interpretation of the feasible set (C.26), (C.29), (C.30) and (C.32). Given 'portions of mass' y^l_{ik} for all $k \in K_l$ are distributed between δ_k and $\sum_{t\in Desc(k_2)} z^l_{ijt}$, according to constraint (C.30), visualized by the upper half of the graph. The lower part of the graph presents the flow of mass from δ_k to $\sum_{t\in T^*_l} z^l_{ijt}$, according to constraint (C.32). Since the problem minimizes a linear combination of z^l_{ijt}, in the optimal solution, all the mass of each y^l_{ik} goes to a single $z^l_{ijt^k_{ij}}$, where t^k_{ij} corresponds the lowest cost coefficient $\hat{c}^l_{ijt}$ among $t \in Desc(k) \cup T^*_l$. See lemma 3 for a more formal treatment.

The feasible set defined by constraints (C.26) to (C.28) can be equivalently expressed by constraints (C.26), (C.29), (C.30) and (C.32). The latter formulation makes it evident that the optimization problem can be viewed as distributing portions of mass y^l_{ik} among z^l_{ijt} for $t \in Desc(k)$ and $t \in T^*_l$. This concept is visualized in figure C.1. In the optimal solution all the mass of y^l_{ik} will go the variable z^l_{ijt} indexed by $t \in Desc(k) \cup T^*_l$ that corresponds to the lowest cost. We define the costs $\hat{c}^l_{ijt} = c^l_{ijt} + \lambda^l_{ijt}$, and denote the optimal costs and indexes for each k as

$$\forall k \in K_l, \quad \tilde{c}_{ijk} = \min_{t\in Desc(k)\cup T^*_l} \hat{c}^l_{ijt}, \quad t^k_{ij} = \arg\min_{t\in Desc(k)\cup T^*_l} \hat{c}^l_{ijt}. \tag{C.33}$$

Since the whole 'mass' of y^l_{ik} goes to the z^l_{ijt} with smallest cost, we claim that $g(\cdot)$ is indeed a linear function, and the coefficient corresponding to y^l_{ik} is $\tilde{c}_{ijk}$, the optimal cost defined above. We note that the for all $k \in K_l$ such that $t^k_{ij} \notin Desc(k)$, the mass from all y^l_{ik} is allocated to a single variable $z^l_{ijt^*_{ij}}$ that has the lowest cost among $t \in T^*_l$. We define

$$\tilde{c}^*_{ij} = \min_{t\in T^*_l} \hat{c}^l_{ijt}, \quad t^*_{ij} = \arg\min_{t\in T^*_l} \hat{c}^l_{ijt}, \tag{C.34}$$

and state our findings formally as lemma 3.

Lemma 3. *The optimal argument $(z^l_{ijt})_{t\in T}$ of the problem (C.25) to (C.28) is*

$$\forall k \in K_l,\ \forall t \in Desc(k) \qquad \begin{cases} z^l_{ijt} = 0 & if \quad t \neq t^k_{ij} \\ z^l_{ijt} = y^l_{ik} & if \quad t = t^k_{ij} \end{cases} \tag{C.35}$$

$$\forall t \in T^*_l \qquad \begin{cases} z^l_{ijt} = 0 & if \quad t \neq t^*_{ij} \\ z^l_{ijt} = \sum\limits_{\substack{k\in K_l \\ s.t. \\ t^k_{ij}=t^*_{ij}}} y^l_{ik} & if \quad t = t^*_{ij} \end{cases} \tag{C.36}$$

Proof. We sketch a proof of the lemma by contradiction.

To prove the first part of (C.35) it is enough to show that any solution where $z^l_{ijt} = \epsilon$ for some $t \neq t^k_{ij}$ and $\epsilon > 0$ is suboptimal, because according to (C.33) the cost $\hat{c}^l_{ijt} > \tilde{c}_{ijk}$. Therefore, we can construct a lower energy solution by increasing the value of $z^l_{ijt^k_{ij}}$ by ϵ and setting $z^l_{ijt} = 0$. The same argument can be used to show that a vector where any of the three remaining statements does not hold is not an optimal solution.

□

We can substitute the z^l_{ijt} in objective (C.25) with optimal values defined in lemma 3 to get

$$g_{ij}(\mathbf{y}^l_i) = \sum_{k\in K_l} \tilde{c}_{ijk} y^l_{ik} \ . \tag{C.37}$$

This function is linear in y^l_{ik}, which concludes the proof that the slave problem has integral vertices.

C.6 Solving the Slave Subproblem

It has been proven that the linear problem of each slave has integral vertices. In consequence, each slave can be seen as a labeling problem where we assign a label $k \in K_l$ to each row i and a label t to each pixel $(i, j) \in \mathcal{I}$. We find the optimal labeling by means of a dynamic program.

Given row class k assigned to row i, it is easy to determine the optimal classes t^k_{ij} for all pixels in the row. Constraint (C.23c) restricts the set of pixel classes that can be used in the row to classes that are descendants of k or to ones that are not descendants of any $k \in K_l$, which we denote T^*_l. The optimal index is

$$t^k_{ij} = \operatorname*{arg\,min}_{t\in Desc(k)\cup T^*_l} (c^l_{ijt} + \lambda^l_{ijt}) \quad . \tag{C.38}$$

From objective (C.22) we derive the optimal cost of assigning row class k to image row i as the sum of costs for each pixel and the per-row cost

$$c^l_{ik} = \sum_j (c^l_{ijt^k_{ij}} + \lambda^l_{ijt^k_{ij}}) + \gamma^l_{ik} \quad . \tag{C.39}$$

Algorithm 4 Dynamic program solving the slave subproblem.

for all $k \in K^l, i \in I$ **do** ▷ dyn. prog. on t_{ij}
  for all $j \in J$ **do**
    $t_{ij}^k \leftarrow \arg\min_{t \in Desc(k) \cup T_l^*} (c_{ijt}^l + \lambda_{ijt}^l)$
  end for
  $c_{ik}^l \leftarrow \sum_j (c_{ijt_{ij}^k}^l + \lambda_{ijt_{ij}^k}^l) + \lambda_{ik}^l$
end for
for all $k \in K$ **do** ▷ dyn. prog. on k_i
  $\phi(1,k) \leftarrow c_{1k}^l$
end for
for $i = 2, \dots h$ **do**
  for $k \in K_l$ **do**
    $\phi(i,k) \leftarrow \min_{k' \in K_l} \phi(i-1,k') + c_{ik}^l + c_{k'k}^l$
    $\kappa(i-1,k) \leftarrow \arg\min_{k' \in K_l} \phi(i-1,k') + c_{ik}^l + c_{k'k}^l$
  end for
end for
$k_i, t_{ij} \leftarrow$ BACKTRACK(ϕ, κ)

The optimal cost of assigning classes for the i first rows, denoted $\phi(i,k)$, where k is the row class assigned to row i, can be recursively defined as

$$\phi(i,k) = \begin{cases} c_{1k}^l & \text{if} \quad i = 1 \\ \min_{k'} \phi(i-1,k') + c_{ik}^l + c_{k'k}^l & \text{otherwise.} \end{cases} \tag{C.40}$$

We use this recursive structure of the subproblem to formulate the Viterbi algorithm 4 for finding its optimal solution. In the first step, the optimal pixel indices t_{ij}^k are determined for each pixel and each $k \in K_l$, according to (C.38). They are then used for determining row class costs c_{ik}^l according to (C.39). Finally the Viterbi recursion of equation (C.40) is used, where $\kappa(i,k)$ stores the optimal class of row i, given that row $i+1$ is given class k. The optimal row and pixel classes are retrieved by backtracking.

Appendix D

Inference in the framework of adjacency patterns

In this appendix we present an algorithm for solving the inference problem posed in chapter 5 in form of equations (5.11-5.15). The following section contains a mathematical derivation, while the next one presents the resulting algorithm.

D.1 Derivation of the inference algorithm

In this section we show how we construct the problem dual to (5.11-5.15).

We introduce two copies of variable $z_{ij\sigma}$, denoted $z'_{ij\sigma}$ and $z''_{ij\sigma}$. We denote the vector of $z'_{ij\sigma}$ for all $(i,j) \in \mathcal{I}$, $\sigma \in S_o$ by $\mathbf{z}'$ and the vector of all $z''_{ij\sigma}$ by $\mathbf{z}''$. We let the new variables, as well as $u_{ij\sigma\sigma'}$ and $v_{ij\sigma\sigma'}$, vary continuously in the interval $[0,1]$. The objective (5.11) can be rewritten as

$$\min_{\mathbf{z}',\mathbf{z}'',\mathbf{u},\mathbf{v}} \sum_{(i,j)\in\mathcal{I}} \sum_{\sigma\in S_o} \frac{1}{2}\phi_{ij\Psi_o(\sigma)}(z'_{ij\sigma} + z''_{ij\sigma}) + \sum_{(i,j)\in\mathcal{I}_v} \sum_{\sigma,\sigma'\in S_o} \theta_o(\sigma,\sigma')v_{ij\sigma\sigma'} + \sum_{(i,j)\in\mathcal{I}_h} \sum_{\sigma,\sigma'\in S_o} \theta_o(\sigma,\sigma')u_{ij\sigma\sigma'} \ , \quad \text{(D.1)}$$

subject to constraints on positivity of the continuous variables

$$\forall (i,j) \in \mathcal{I}, \quad \sigma \in S_o, \quad z'_{ij\sigma} \geq 0, \quad \text{(D.2a)}$$
$$\forall (i,j) \in \mathcal{I}, \quad \sigma \in S_o, \quad z''_{ij\sigma} \geq 0, \quad \text{(D.2b)}$$
$$\forall (i,j) \in \mathcal{I}_v, \sigma, \sigma' \in S_o, \quad v_{ij\sigma\sigma'} \geq 0, \quad \text{(D.2c)}$$
$$\forall (i,j) \in \mathcal{I}_h, \sigma, \sigma' \in S_o, \quad u_{ij\sigma\sigma'} \geq 0, \quad \text{(D.2d)}$$

the simplex constraints (5.12) reformulated on the new variables

$$\forall (i,j) \in \mathcal{I}, \quad \sum_{\sigma\in S_o} z'_{ij\sigma} = 1 \ , \quad \text{(D.3a)}$$

$$\forall (i,j) \in \mathcal{I}, \quad \sum_{\sigma\in S_o} z''_{ij\sigma} = 1 \ , \quad \text{(D.3b)}$$

coupling constraints for $z'_{ij\sigma}$ and $z''_{ij\sigma}$,

$$\forall (i,j) \in \mathcal{I}, \sigma \in S_o, \quad z'_{ij\sigma} = z''_{ij\sigma} \ , \tag{D.4}$$

and a reformulation of constraints (5.13), (5.14), (5.15a) and (5.15b) on $z'_{ij\sigma}$ and $z''_{ij\sigma}$ that we present below. Given equation (D.4), each of constraints (5.13), (5.14), (5.15a) and (5.15b) can be defined on just one of the new variables, that is, either on $z'_{ij\sigma}$, or on $z''_{ij\sigma}$. To obtain a formulation which decomposes into horizontal and vertical chains, we require $z'_{ij\sigma}$ to satisfy the constraints on classes of vertical neighbors, and require $z''_{ij\sigma}$ to satisfy the horizontal constraints

$$\forall (i,j) \in \mathcal{I}_v, \ \forall \sigma \in S_o, \qquad \sum_{\sigma' \in S_o} v_{ij\sigma\sigma'} = z'_{ij\sigma}, \qquad \sum_{\sigma' \in S_o} v_{ij\sigma'\sigma} = z'_{i+1\,j\sigma'}, \tag{D.5a}$$

$$\forall (i,j) \in \mathcal{I}_h, \ \forall \sigma \in S_o, \qquad \sum_{\sigma' \in S_o} u_{ij\sigma\sigma'} = z''_{ij\sigma}, \qquad \sum_{\sigma' \in S_o} u_{ij\sigma'\sigma} = z''_{ij+1\sigma'}, \tag{D.5b}$$

$$\forall (i,j) \in \mathcal{I}_v, \forall (\sigma,\sigma') \notin V_o, \quad v_{ij\sigma\sigma'} = 0 \ , \tag{D.6a}$$

$$\forall (i,j) \in \mathcal{I}_h, \forall (\sigma,\sigma') \notin H_o, \quad u_{ij\sigma\sigma'} = 0 \ . \tag{D.6b}$$

It is obvious that, when constraint (D.4) is satisfied, objective (D.1) is equivalent to the original objective (5.11).

We define the feasible set C' as the set of vectors $(\mathbf{z}, \mathbf{v})$ that satisfy the vertical constraints (D.2a), (D.2c), (D.3a), (D.5a), (D.6a) and the set C'' as the set of vectors $(\mathbf{z}, \mathbf{u})$ satisfying the horizontal constraints (D.2b), (D.2d), (D.3b), (D.5b), (D.6b). We construct a Lagrangian for problem (D.1) with respect to constraint (D.4)

$$\begin{aligned} L_D(\boldsymbol{\lambda}) = \min_{\substack{(\mathbf{z}',\mathbf{v}) \in C' \\ (\mathbf{z}'',\mathbf{u}) \in C''}} & \sum_{(i,j) \in \mathcal{I}} \sum_{\sigma \in S_o} \frac{1}{2} \phi_{ij\Psi_o(\sigma)} (z'_{ij\sigma} + z''_{ij\sigma}) + \\ & \sum_{(i,j) \in \mathcal{I}_v} \sum_{\sigma,\sigma' \in S_o} \theta_o(\sigma,\sigma') v_{ij\sigma\sigma'} + \sum_{(i,j) \in \mathcal{I}_h} \sum_{\sigma,\sigma' \in S_o} \theta_o(\sigma,\sigma') u_{ij\sigma\sigma'} + \\ & \sum_{(i,j) \in \mathcal{I}} \sum_{\sigma \in S_o} \lambda_{ij\sigma} (z'_{ij\sigma} - z''_{ij\sigma}) \ . \end{aligned} \tag{D.7}$$

where $\boldsymbol{\lambda}$ is a vector of Lagrange multipliers $\lambda_{ij\sigma}$ for all $(i,j) \in \mathcal{I}$ and all $\sigma \in S_o$.

We formulate a dual problem

$$\max_{\boldsymbol{\lambda}} L_D(\boldsymbol{\lambda}) \ . \tag{D.8}$$

We solve the dual using a subgradient ascent scheme. In each iteration, we calculate the subgradient of the objective and update $\boldsymbol{\lambda}$ by making a step in the direction of the subgradient. We use a fixed sequence of decaying step sizes. It can be shown (see [26] for details) that the subdifferential of the objective contains the vector $\tilde{\mathbf{z}} = \hat{\mathbf{z}}' - \hat{\mathbf{z}}''$, where the pair $(\hat{\mathbf{z}}', \hat{\mathbf{z}}'')$ minimizes (D.7). However, the variables are not coupled by any constraints and so we have

$$\hat{\mathbf{z}}' = \operatorname*{arg\,min}_{(\mathbf{z}',\mathbf{v}) \in C'} \sum_{(i,j) \in \mathcal{I}} \sum_{\sigma \in S_o} (\frac{1}{2} \phi_{ij\Psi_o(\sigma)} + \lambda_{ij\sigma}) z'_{ij\sigma} + \sum_{(i,j) \in \mathcal{I}_v} \sum_{\sigma,\sigma' \in S_o} \theta_o(\sigma,\sigma') v_{ij\sigma\sigma'} \tag{D.9a}$$

and

$$\hat{\mathbf{z}}'' = \underset{(\mathbf{z}'',\mathbf{u})\in C''}{\arg\min} \sum_{(i,j)\in\mathcal{I}} \sum_{\sigma\in S_o} (\frac{1}{2}\phi_{ij\Psi_o(\sigma)} - \lambda_{ij\sigma}) z''_{ij\sigma} + \sum_{(i,j)\in\mathcal{I}_h} \sum_{\sigma,\sigma'\in S_o} \theta_o(\sigma,\sigma') u_{ij\sigma\sigma'} \ . \quad \text{(D.9b)}$$

The minimizations (D.9) are equivalent to finding the most likely configurations of independent Markov chains over image rows and columns, with hard constraints on neighboring labels, and can be solved by running the Viterbi algorithm independently on each row and column.

D.2 The inference algorithm

The resulting algorithm is presented in algorithm (5), where superscript n denotes the iteration number, α^n is the step size in n-th iteration, $\mathbf{z}_{\cdot j}$, $\boldsymbol{\phi}_{\cdot j}$ and $\boldsymbol{\lambda}_{\cdot j}$ are vectors of variables and costs corresponding to image column j and $\mathbf{z}_{i\cdot}$, $\boldsymbol{\phi}_{i\cdot}$ and $\boldsymbol{\lambda}_{i\cdot}$ are the vectors corresponding to row i. By $\mathring{z}_{ij\sigma}$ we denote the number of times class σ has been assigned to pixel (i,j) during the operation of the algorithm. We denote the vector of all $\mathring{z}_{ij\sigma}$ by $\mathring{\mathbf{z}}$.

Algorithm 5 Dual Decomposition on a 4-connected grid with slaves solving Markov Chains

$\boldsymbol{\lambda}^0 \leftarrow 0$
$n \leftarrow 1$
$\mathring{\mathbf{z}} \leftarrow 0$
while not converged **do**
  $\forall j \in J \quad \hat{\mathbf{z}}'^n_{\cdot j} \leftarrow \textsc{Viterbi}(\frac{1}{2}\boldsymbol{\phi}_{\cdot j} + \boldsymbol{\lambda}^{n-1}_{\cdot j}, V)$
  $\forall i \in I \quad \hat{\mathbf{z}}''^n_{i\cdot} \leftarrow \textsc{Viterbi}(\frac{1}{2}\boldsymbol{\phi}_{i\cdot} - \boldsymbol{\lambda}^{n-1}_{i\cdot}, H)$
  $\boldsymbol{\lambda}^n \leftarrow \boldsymbol{\lambda}^{n-1} + \frac{\alpha^n}{2}(\hat{\mathbf{z}}'^n - \hat{\mathbf{z}}''^n)$
  $\mathring{\mathbf{z}} \leftarrow \mathring{\mathbf{z}} + \hat{\mathbf{z}}'^n + \hat{\mathbf{z}}''^n$
  $n \leftarrow n + 1$
end while
$\hat{\mathbf{z}} \leftarrow \textsc{GetFinalZ}(\mathring{\mathbf{z}}, V, H)$

After a number of iterations, the variables $\mathbf{z}'$ and $\mathbf{z}''$ agree on the labels for most pixels. Heuristics can then be used to extract labels for the pixels for which $\mathbf{z}'$ and $\mathbf{z}''$ disagree [26]. In algorithm 5, we denote this procedure by GetFinalZ. One possible method is to count how many times a pixel has been assigned each of the possible labels, by accumulating the label frequency $\mathring{\mathbf{z}}$ over a number of iterations, and selecting for each pixel the most frequent label

$$\forall (i,j) \in \mathcal{I}, \ \hat{z}_{ij\sigma} = \begin{cases} 1 & \text{if} \quad \sigma = \arg\max_{\sigma'} \mathring{z}_{ij\sigma'} \\ 0 & \text{otherwise.} \end{cases} \quad \text{(D.10)}$$

However, a solution obtained in this manner would not necessarily be consistent with the constraints on classes of adjacent pixels. To extract a solution consistent with the constraints, we modify this heuristic. We present the concept in algorithm

6. For a randomly drawn row (or column) index $\hat{i}$, we extract its labeling using the Viterbi algorithm with costs corresponding to label frequencies. In algorithm 6, this subprocedure is represented as ViterbiMax. It finds the labeling of the row that maximizes the total frequency of the labels assigned to pixels in the row, but only assigns to pairs of horizontally neighboring pixels pairs of labels that belong to H. Then we repeat the procedure for the neighboring row. In algorithm 6, this is represented as ViterbiMaxWithConstraints, which uses the sets of allowed pairs of vertically adjacent classes V and the labels of the previous row to constrain the sets of labels that can be assigned to each pixel in the next row. The process is repeated for consecutive rows until a complete labeling is extracted. We sample a number of labelings initializing the procedure on randomly selected rows and columns, and keep the one with the lowest energy as the final solution.

Algorithm 6 Heuristics used to extract a labeling that satisfies hard constraints on classes of adjacent pixels. The solution is initialized with row $\hat{i}$ of the image. (The same operation can be performed for image columns, given an initial column index $\hat{j}$.)

$\hat{\mathbf{z}}_{\hat{i}\cdot} \leftarrow$ ViterbiMax$(\mathring{\mathbf{z}}_{\hat{i}\cdot}, H)$
for $i = \hat{i} + 1$ to h **do**
  $\hat{\mathbf{z}}_{i\cdot} \leftarrow$ ViterbiMaxWithConstraints$(\mathring{\mathbf{z}}_{i\cdot}, H, \hat{\mathbf{z}}_{i-1\cdot}, V)$
end for
for $i = \hat{i} - 1$ to 1 **do**
  $\hat{\mathbf{z}}_{i\cdot} \leftarrow$ ViterbiMaxWithConstraints$(\mathring{\mathbf{z}}_{i\cdot}, H, \hat{\mathbf{z}}_{i+1\cdot}, V)$
end for

We remark on the theoretical possibility of designing a prior for which algorithm 6 could fail to find a labeling consistent with the constraints on classes of adjacent pixels. Such a situation could emerge if there exists a row (or column) labeling for which the neighboring row (resp. column) can not be labeled in such a way that the pairs of vertically and horizontally neighboring labels belong to V and H. Since row and column labelings consistent with a prior are defined in terms of constraints on classes of pairs of adjacent pixels, the formal condition for a prior for which algorithm 6 is guaranteed to find a labeling consistent with the constraints can be formulated in terms of pairs of allowed classes of adjacent pixels

$$\forall(\sigma_1, \sigma_2) \in V_o,\ \exists(\sigma_1', \sigma_2') \in V_o,\ \text{s.t. } (\sigma_1, \sigma_1') \in H_o, (\sigma_2, \sigma_2') \in H_o\ , \tag{D.11a}$$
$$\forall(\sigma_1, \sigma_2) \in H_o, \exists(\sigma_1', \sigma_2') \in H_o, \text{ s.t. } (\sigma_1, \sigma_1') \in V_o,\ (\sigma_2, \sigma_2') \in V_o\ . \tag{D.11b}$$

The adjacency patterns for which algorithm 6 can fail to find a labeling satisfying the constraints, in practice do not encode more useful priors than ones that satisfy conditions (D.11). In particular, for any prior for which $\forall s \in S,\ (s, s) \in V$ and $(s, s) \in H$, which corresponds o the most practical case, when a facade element can have an arbitrary size, algorithm 6 always finds a valid labeling. The same holds for adjacency patterns obtained by transforming grid patterns according to (5.1), and the alphabet of irregular shapes presented in section 5.3.2. Finally, algorithm 6 can easily be modified to detect a situation when it fails to generate a labeling that respects all the constraints.

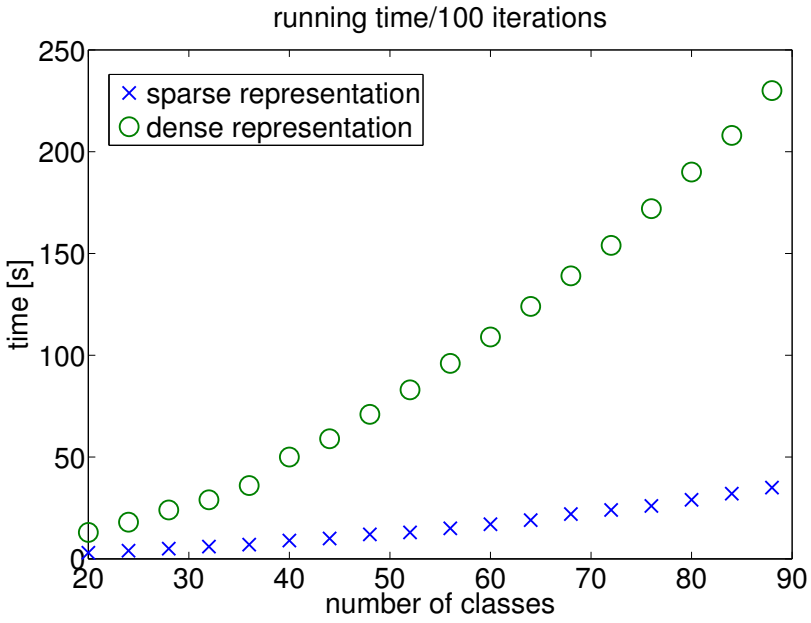


Figure D.1: The running time with respect to number of (pre-semantic) classes for the implementation based on sparse and dense representation of a table of allowed neighbor classes. The displayed time contains 100 iterations of the dual decomposition algorithm and extraction of the primal solution. The experiment has been performed on a single image of the ECP dataset. We used a 6-core Core-i7 processor with 3GHz clock.

The complexity of the Viterbi algorithm used to solve the subproblems can be decreased by exploiting the fact that only some pairs of neighboring classes are allowed. In each step, the algorithm determines the optimal class of the previously processed pixel, given the class of the current one. We can speed up the process by only checking the previous classes that constitute valid pairs with the current class. This is achieved by storing the matrices of allowed classes of neighboring pixels in sparse form. We compare the dependence of the running time on the number of classes for the sparse and dense representations and present the results in figure D.1. We vary the number of classes by starting with a complex prior with many classes and then randomly removing classes. The number of classes is a good measure of complexity of shapes represented by a prior. Exploiting the sparsity in the implementation of the Viterbi algorithm results in a speedup of roughly 5 times in this particular example.

Bibliography

[1] Ahuja, N. and Todorovic, S. (2008). Connected Segmentation Tree - A Joint Representation of Region Layout and Hierarchy. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*.

[2] Bai, J., Song, Q., Veksler, O., and Wu, X. (2012). Fast dynamic programming for labeling problems with ordering constraints. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1728–1735.

[3] Barinova, O., Lempitsky, V., Tretiak, E., and Kohli, P. (2010). Geometric image parsing in man-made environments. In *Proc. European Conference on Computer Vision (ECCV)*.

[4] Berg, A. C., Grabler, F., and Malik, J. (2007). Parsing images of architectural scenes. In *Proc. International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE.

[5] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. In *Foundations and Trends in Machine Learning*, volume 3, pages 1–122.

[6] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *Transactions on Pattern Analysis and Machine Intelligence*, 23:1222–1239.

[7] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[8] Chib, S. and Greenberg, E. (1995). Understanding the Metropolis-Hastings algorithm. *The American Statistician*, 49(4):327–335.

[9] Cohen, A., Schwing, A., and Pollefeys, M. (2014). Efficient structured parsing of facades using dynamic programming. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[10] Dai, D., Prasad, M., Schmitt, G., and Van Gool, L. (2012). Learning domain knowledge for facade labeling. In *Proc. European Conference on Computer Vision (ECCV)*.

[11] Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79.

[12] Felzenszwalb, P. F. and Veksler, O. (2010). Tiered scene labeling with dynamic programming. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3097–3104.

[13] Fischler, M. A. and Elschlager, R. A. (1973). The Representation and Matching of Pictorial Structures. *Computers, Transactions on*, C-22(1):67–92.

[14] Forney Jr, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

[15] Furst, L., Mernik, M., and Mahnic, V. (2011). Improving the graph grammar parser of Rekers and Schürr. *IET Software*, pages 246–261.

[16] Gadde, R., Marlet, R., and Paragios, N. (2014). Learning grammars for architecture-specific facade parsing. Research Report RR-8600, Ecode des Ponts ParisTECH.

[17] Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. A. (2011). Object detection with grammar models. In *NIPS*, pages 442–450.

[18] Han, F. and Zhu, S.-C. (2009). Bottom-up/top-down image parsing with attribute graph grammar. *Transactions on Pattern Analysis and Machine Intelligence*, 31(1):59–73.

[19] Heitz, G. and Koller, D. (2008). Learning spatial context: Using stuff to find things. In *Proc. 10th European Conference on Computer Vision (ECCV)*.

[20] Hernandez, J. and Marcotegui, B. (2009). Morphological segmentation of building facade images. In *Proceedings of the 16th IEEE international conference on Image processing*, ICIP'09, pages 3977–3980, Piscataway, NJ, USA. IEEE Press.

[21] Jin, Y. and Geman, S. (2006). Context and hierarchy in a probabilistic image model. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2145–2152. IEEE.

[22] Kappes, J. H., Andres, B., Hamprecht, F. A., Schnorr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B. X., Lellmann, J., Komodakis, N., et al. (2013). A comparative study of modern inference techniques for discrete energy minimization problems. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1328–1335. IEEE.

[23] Klempien-Hinrichs, R. (2002). Context-free hypergraph grammars with node rewriting. *Electronic Notes in Theoretical Computer Science*, 51:202 – 211.

[24] Kolmogorov, V. (2006). Convergent Tree-Reweighted message passing for energy minimization. *Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583.

[25] Komodakis, N. (2011). Efficient training for pairwise or higher order CRFs via dual decomposition. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1841–1848.

[26] Komodakis, N., Paragios, N., and Tziritas, G. (2011). MRF energy minimization and beyond via dual decomposition. *Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552.

[27] Korč, F. and Förstner, W. (2009). eTRIMS Image Database for interpreting images of man-made scenes. Technical Report TR-IGG-P-2009-01, University of Bonn.

[28] Koutsourakis, P., Simon, L., Teboul, O., Tziritas, G., and Paragios, N. (2009). Single view reconstruction using shape grammars for urban environments. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1795–1802. IEEE.

[29] Ladický, L., Russell, C., Kohli, P., and Torr, P. H. S. (2013). Associative hierarchical random fields. *Transactions on Pattern Analysis and Machine Intelligence*, 99(PrePrints).

[30] Leibe, B., Leonardis, A., and Schiele, B. (2008). Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vision*, 77(1-3):259–289.

[31] Liu, X., Veksler, O., and Samarabandu, J. (2010). Order-preserving moves for graph-cut-based optimization. *Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1182–1196.

[32] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

[33] Martinovic, A., Mathias, M., Weissenberg, J., and Van Gool, L. (2012). A three-layered approach to facade parsing. In *Proc. European Conference on Computer Vision (ECCV)*, pages 416–429. Springer.

[34] Martinovic, A. and Van Gool, L. (2013). Bayesian grammar learning for inverse procedural modeling. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[35] Martinovic, A. and Van Gool, L. (2014). Hierarchical co-segmentation of building facades. In *2nd International Conference on 3D Vision (3DV)*, volume 1, pages 409–416.

[36] Mathias, M., Martinovic, A., Weissenberg, J., and Van Gool, L. (2011). Procedural 3D building reconstruction using shape grammars and detectors. In *Joint Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*.

[37] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., and Van Gool, L. (2005). A comparison of affine region detectors. *International Journal of Computer Vision*, 65:2005.

[38] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2006). Procedural modeling of buildings. *ACM Transations on Graphics*, 25(3):614–623.

[39] Müller, P., Zeng, G., Wonka, P., and Van Gool, L. (2007). Image-based procedural modeling of facades. *ACM. Transactions on Graphics*, 26(3):85.

[40] Murphy, K. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division.

[41] Musialski, P., Wimmer, M., and Wonka, P. (2012). Interactive coherence-based façade modeling. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2012)*, 31:661–670.

[42] Ohta, Y., Kanade, T., and Sakai, T. (1978). An analysis system for scenes containing objects with substructures. In *Proceedings of the Fourth International Joint Conference on Pattern Recognitions*, pages 752–754.

[43] Ohta, Y., Kanade, T., and Sakai, T. (1979). A production system for region analysis. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 684 – 686.

[44] Ok, D., Kozinski, M., Marlet, R., and Paragios, N. (2012). High-level bottom-up cues for top-down parsing of facade images. In *2nd Joint Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*.

[45] Raguram, R., Frahm, J.-M., and Pollefeys, M. (2008). A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In *Proc. European Conference on Computer Vision (ECCV)*, pages 500–513.

[46] Rekers, J. and Schürr, A. (1995). A parsing algorithm for context-sensitive graph grammars. Technical Report 95-05, Leiden University.

[47] Riemenschneider, H., Krispel, U., Thaller, W., Donoser, M., Havemann, S., Fellner, D., and Bischof, H. (2012). Irregular lattices for complex shape grammar facade parsing. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[48] Roy, A. and Todorovic, S. (2014). Scene labeling using beam search under mutex constraints. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1178–1185.

[49] Rozenberg, G., editor (1997). *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing Co., Inc.

[50] Shotton, J., Winn, J. M., Rother, C., and Criminisi, A. (2006). *TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proc. European Conference on Computer Vision (ECCV)*.

[51] Si, Z. and Zhu, S.-C. (2013). Learning and-or templates for object recognition and detection. *Transactions on Pattern Analysis and Machine Intelligence*, 99(PrePrints):1.

[52] Simon, L., Teboul, O., Koutsourakis, P., Van Gool, L., and Paragios, N. (2012). Parameter-free/Pareto-driven procedural 3D reconstruction of buildings from ground-level sequences. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[53] Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)*.

[54] Sontag, D., Globerson, A., and Jaakkola, T. (2011). Introduction to dual decomposition for inference. In Sra, S., Nowozin, S., and Wright, S. J., editors, *Optimization for Machine Learning*. MIT Press.

[55] Stiny, G. N. (1975). *Pictorial and Formal Aspects of Shape and Shape Grammars and Aesthetic Systems*. PhD thesis, University of California, Los Angeles. AAI7526993.

[56] Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2008). A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. *Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080.

[57] Tarlow, D., Batra, D., Kohli, P., and Kolmogorov, V. (2011). Dynamic tree block coordinate ascent. In *International Conference on Machine Learning (ICML)*, pages 113–120.

[58] Teboul, O. (2011). *Shape Grammar Parsing : Application to Image-based Modeling*. PhD thesis, Ecole Centrale Paris.

[59] Teboul, O., Kokkinos, I., Simon, L., Koutsourakis, P., and Paragios, N. (2011). Shape grammar parsing via reinforcement learning. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2273–2280. IEEE.

[60] Teboul, O., Simon, L., Koutsourakis, P., and Paragios, N. (2010). Segmentation of building facades using procedural shape priors. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3105–3112. IEEE.

[61] Vermeulen, J. T. (1996). Viability of a parsing algorithm for context-sensitive graph grammars. Technical report, Leiden Institute of Advanced Computer Science.

[62] Viola, P. A. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154.

[63] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, Transactions on*, 13(2):260–269.

[64] von Gioi, R. G., Jakubowicz, J., Morel, J.-M., and Randall, G. (2010). LSD: A fast line segment detector with a false detection control. *Transactions on Pattern Analysis and Machine Intelligence*, 32:722–732.

[65] Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):305.

[66] Wang, W., Pollak, I., Wong, T.-S., Bouman, C. A., and Harper, M. P. (2006). Hierarchical stochastic image grammars for classification and segmentation. *Transactions on Image Processing*, 15:3033–3052.

[67] Weissenberg, J., Riemenschneider, H., Prasad, M., and Van Gool, L. (2013). Is there a procedural logic to architecture? In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 185–192. IEEE.

[68] Werner, T. (2007). A linear programming approach to max-sum problem: A review. *Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179.

[69] Wu, F., Yan, D.-M., Dong, W., Zhang, X., and Wonka, P. (2014). Inverse procedural modeling of facade layouts. *ACM Transactions on Graphics*, 33(4):121:1–121:10.

[70] Yang, C., Han, T., Quan, L., and Tai, C.-L. (2012). Parsing facade with rank-one approximation. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1720–1727. IEEE.

[71] Yang, Y. and Ramanan, D. (2011). Articulated pose estimation with flexible mixtures-of-parts. In *Proc. Internation Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[72] Zhao, P., Fang, T., Xiao, J., Zhang, H., Zhao, Q., and Quan, L. (2010). Rectilinear parsing of architecture in urban environment. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 342–349. IEEE.

[73] Zhu, S.-C. and Mumford, D. (2006). A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Visions*, 2(4):259–362.