Méthodologie de conception d'un système expert pour la généralisation cartographique

Xiao Chun Zhao Épouse Boury

Thèse de Doctorat de l'Ecole Nationale des Ponts et Chaussées

Spécialité:

Informatique

présentée

par ZHAO Xiao Chun épouse BOURY
pour obtenir le titre de:

Docteur de l'Ecole Nationale des Ponts et Chaussées

sujet:

Méthodologie de Conception d'un Système Expert pour la Généralisation Cartographique

soutenue le 14 novembre 1990
devant le jury composé de:

Monsieur	Georges	STAMON	Président
Monsieur	Jean-Paul	HATON	Rapporteur
Monsieur	Bruno	PASQUIER	Rapporteur
Monsieur	Hervé	LE MEN	Examinateur
Monsieur	René	LALEMENT	Examinateur

ECOLE NATIONALE DES PONTS ET CHAUSSEES

REMERCIEMENTS

Cette thèse est le fruit d'une coopération entre l'Ecole Nationale des Ponts et Chaussées et l'Institut Géographique National.

Je voudrais faire part de ma reconnaissance à Messieurs N. Bouleau au CERMA-ENPC, H. Le Men au COGIT-IGN, et R. Lalement au CERMICS-ENPC, pour l'aide précieuse qu'il m'ont bien accordée tout au long de cette thèse en en ont assurant la direction, et en m'accueillant chaleureusement dans leur laboratoire.

J'exprime toute ma gratitude à Monsieur G. Stamon de m'avoir fait l'honneur de présider le jury, à Messieurs J.P. Haton et B. Pasquier d'avoir bien voulu accepter d'être rapporteurs de ce travail, malgré leur surcharge de travail.

Pour les conseils, les marques d'intérêt qu'ils ont portés à mon travail en de nombreuses occasions et les corrections de fautes de français à la lecture de cette thèse, je remercie Messieurs G. Weger, J.G. Affholder, R. Lalement, G. Caplain.

Mes remerciements vont aussi à tous mes collègues du COGIT, du CERMA et du GISE, notamment Mesdames V. Serre, M. Razurel, E. Frugier, Messieurs O. Jamet, La Rivière, pour une ambiance amicale du travail, et à tous ceux qui m'ont apporté leur soutien.

Je suis très reconnaissante envers ma famille en Chine, mon époux Pierre et toute ma belle-famille pour leur attention spéciale et leur soutien moral infiniment important au cours de ce travail.

目录 : table des matières

1 Introduction 9
- 1.1 Aperçu de la généralisation cartographique . . . 10
- 1.2 Difficultés essentielles . . . 11
- 1.3 Eléments d'intelligence artificielle . . . 13
- 1.4 Perspectives . . . 16

I Cartographie 19

2 La généralisation cartographique 21
- 2.1 Analyse du problème . . . 21
- 2.2 Cas des voies de communication . . . 23
  - 2.2.1 Carrefour . . . 23
  - 2.2.2 Voies . . . 23
- 2.3 Les facteurs d'influence . . . 24
- 2.4 Règles de lisibilité . . . 25
- 2.5 Problème des signes conventionnels . . . 27
  - 2.5.1 Nécessité des signes conventionnels . . . 27
  - 2.5.2 Les signes conventionnels linéaires . . . 27
  - 2.5.3 Evolution des signes avec l'échelle . . . 29
- 2.6 Placement des toponymes . . . 31
  - 2.6.1 Ponctuel . . . 33
  - 2.6.2 Linéaire . . . 33
- 2.7 Un critère de résolution de problème . . . 34

II Outils Informatiques 39

3 Choix de modes de représentation des connaissances 41
- 3.1 Caractéristiques des systèmes experts . . . 41
  - 3.1.1 Composants de base d'un système expert . . . 41
  - 3.1.2 Notre système, en résumé . . . 44
- 3.2 Difficulté de concevoir la connaissance . . . 45
- 3.3 Quelques modes de représentation des connaissances . . . 49
- 3.4 Procédural ou déclaratif? . . . 50
- 3.5 Raisonner logiquement . . . 52
  - 3.5.1 Règles d'inférence en logique . . . 53

3.5.2 La logique des prédicats 54

4 Les langages orientés objets **56**
4.1 Les réseaux sémantiques 58
4.2 Les prototypes 61
4.3 Les classes et objets 61
4.4 Les frames 63
4.5 Les acteurs 64
4.6 Les langages hybrides 64
4.7 Les messages 64
4.8 L'héritage 65
4.9 La relation d'héritage 68
4.10 Objet et représentation des connaissance 69
4.11 Les exceptions 70
4.11.1 Héritage et exception 70
4.11.2 Chemins d'héritabilité 71
4.11.3 Quelques techniques de résolution 72
4.12 L'intelligibilité 73
4.13 Les défauts 73
4.14 Comparaison des formalismes 74

5 Lisp et objets **76**
5.1 Intérêt de programmer en Lisp 76
5.2 Ceyx 77
5.3 Common Lisp Object System: CLOS 85
5.4 KEE 87

III Réalisation **89**

6 Modélisation des opérations graphiques **91**
6.1 Les mesures 92
6.2 Règles de sélection 92
6.2.1 Cas général 92
6.2.2 Idées théoriques de sélection 94
6.2.3 Cas linéaire 97
6.3 Règles de schématisation structurale 98
6.4 Signes linéaires 101
6.4.1 Définitions des segments, arcs, tronçons 101
6.4.2 Définition de la convexité 102
6.4.3 Règles globales de schématisation 103
6.4.4 Deux segments non alignés: virage 104
6.4.5 Linéarisation locale 106
6.5 Règles d'harmonisation 110
6.5.1 Hiérarchie des éléments 111
6.5.2 Séparer deux signes voisins ponctuels 113
6.5.3 Séparer un groupe de signes ponctuels 114
6.5.4 Séparer deux signes voisins linéaires 117
6.5.5 Séparer un groupe de signes voisins linéaires 118

6.5.6 Translation d'un signe linéaire 118
6.6 Traitement des carrefours 120
6.6.1 Importance de carrefours 120
6.6.2 Domaine de convexité d'un carrefour 121
6.6.3 Distance entre carrefours 125
6.6.4 Direction de déplacement d'un carrefour 127
6.7 Stratégie générale 128
6.8 Résumé 128

7 La base de connaissances 130
7.1 Les structures des objets 130
7.1.1 Organisation des objets 131
7.1.2 Les objets composites 138
7.2 Les procédures des objets 139
7.3 Exemples: les sommets-carrefours 140
7.4 Formuler les connaissances 141
7.4.1 Formuler les faits 141
7.4.2 Les prédicats 142
7.4.3 Formuler les règles 143
7.4.4 L'organisation des règles 147

8 Un petit moteur d'inférences 151
8.1 Le Filtrage et l'unification 151
8.2 Avantages et inconvénients 153
8.3 Résolution 154
8.4 Chaînage avant 156
8.5 Mode d'inférence 157
8.6 Stratégie du développement de la recherche 157
8.7 Non monotonie 157
8.8 Fonctionnement sur un exemple 158

9 Exemples 162
9.1 La base des objets du réseau 162
9.2 Processus de généralisation 163
9.3 De 1:25 000 aux échelles inférieures 164
9.3.1 De 1:25 000 à 1:50 000 164
9.3.2 De 1:25 000 à 1:100 000 165
9.3.3 De 1:25 000 à 1:250 000 167
9.3.4 De 1:25 000 à 1:500 000 169
9.3.5 De 1:25 000 à 1:1000 000 169

IV Conclusion générale 181

V Annexe 187

A Code des Programmes **189**
A.1 Structuration et gestion interne des objets 189
A.1.1 La représentation interne des faits 189
A.1.2 La représentation interne des règles 191
A.2 Le filtrage 196
A.3 L'unification 202
A.4 Code du moteur d'inférence 205

B Art de PostScript **212**
B.1 Introduction 212
B.2 PostScript et la cartographie 213
B.2.1 Modèles d'image 213
B.2.2 Style de programmation 214
B.3 Exemples de constructions des graphiques 214

C Fichiers des règles **219**

Bibliographie **229**

: Liste des Figures

1.1 Un petit exemple de carrefour à généraliser 12
1.2 Système essentiel de G.C. 17

2.1 Positions préférentielles 33
2.2 Les directions de placement de toponymes 34
2.3 Réduction de carte à la façon photocopieuse 36
2.4 La carte de la ville Marmande: les réductions en prenant les signes conventionnels correspondant sans généralisation 37

4.1 Les champs de l'objet de **voie routière** 57
4.2 Réseau communication 58
4.3 Le lien ATPG et le lien ATPS 59
4.4 Réseau à l'échelle E= 1:250 000 60
4.5 Les sous-classes de la classe de **réseau de communication** 66
4.6 Graphe d'héritage, exception au niveau de **Rivière** 70
4.7 Une exception au niveau de **Canal** 71

5.1 La hiérarchie des *packages* 77

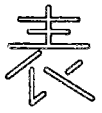
6.1 Axes de voies de communication 99
6.2 Quand l'échelle réduit. 100
6.3 Segment, arc, tronçon, route 101
6.4 Enveloppe convexe de 3 points 102
6.5 Les 2 domaines de convexité de 2 tronçons 103
6.6 2 segments se croisent 104
6.7 Arrondir les coins des segments. 107
6.8 Petits segments à supprimer. 107
6.9 Petites sinuosités pouvant être supprimées. 108
6.10 Superposition de deux signes linéaires 110
6.11 Séparation de deux éléments ponctuels 113
6.12 Expansion par rapport au centre géométrique 115
6.13 Expansion par rapport au barycentre 115
6.14 Homothétie d'un groupe d'éléments ponctuels 115
6.15 Homothétie avec une distribution exponentielle 117
6.16 Translation globale sans contrainte 117
6.17 Translation locale + linéarisation locale 118
6.18 Translation avec contraintes: avant linéarisation 119
6.19 Translation avec contraintes: après linéarisation 120

6.20 Importance de carrefour 121
6.21 Convexité d'un carrefour 122
6.22 Distance de 2 carrefours non liés à leur axe 126
6.23 Distance de 2 carrefours reliés à l'axe 127
6.24 Stratégie de résolution 129

7.1 Les objets du réseau de communication 134
7.2 Les compositions des objets du réseau 136
7.3 Les classes et sous-classes de **Voies de communication** 137
7.4 Structuration des règles 147
7.5 Toutes les règles de la G.C. 150

9.1 De 1:25 000 à 1:50 000 sans généralisation 170
9.2 De 1:25 000 à 1:50 000 avec généralisation 171
9.3 De 1:25 000 à 1:100 000 sans généralisation 172
9.4 De 1:25 000 à 1:100 000 avec généralisation 173
9.5 De 1:25 000 à 1:250 000 sans généralisation 174
9.6 De 1:25 000 à 1:250 000 avec généralisation 175
9.7 De 1:25 000 à 1:500 000 sans généralisation 176
9.8 De 1:25 000 à 1:500 000 avec généralisation 177
9.9 De 1:25 000 à 1:1000 000 sans généralisation 178
9.10 De 1:25 000 à 1:1000 000 avec généralisation 179

B.1 Tracer des routes avec PostScript 216



: Liste des Tableaux

2.1 Comparer une *carte* avec une *parole* 22
2.2 Tableau de dimensions graphiques minimales 26
2.3 Evolution des signes conventionnels avec l'échelle 28
2.4 Tableau de comparaison des signes conventionnels 32

3.1 Les variables de la cartographie 46

6.1 Tableau de pourcentages de détails conservés 96
6.2 Tableau de relation de sélection utilisable pour les échelles 98

7.1 Ajout d'un fait 148
7.2 Suppression d'un fait 149

Chapitre 1

Introduction

La carte
est la représentation conventionnelle, généralement plane, en positions relatives, des phénomènes concrets ou abstraits localisables dans l'espace.

CFC[1]

La cartographie
est l'ensemble des études et des opérations scientifiques, artistiques et techniques intervenant à partir des résultats d'observations directes ou de l'exploitation d'une documentation, en vue de l'élaboration de cartes, plans et autres modes d'expression, ainsi que dans leur utilisation.

ACI[2] 1966

L'intelligence
est la faculté de comprendre, de saisir par la pensée; l'aptitude à s'adapter à une situation, à choisir en fonction des circonstances; la capacité de donner un sens à telle ou telle chose.

L'intelligence artificielle
est l'intelligence humaine simulée par les machines.

Petit Larousse édition 89

La généralisation cartographique (GC) joue un rôle important dans les systèmes de production des cartes assistée par ordinateur et dans les systèmes d'informations géographiques. Elle est depuis longtemps l'enjeu de la cartographie en raison de sa complexité intrinsèque, et on ne connaît pas de méthodes de résolution directes et assurées. Devant un problème complexe comme la GC, il est difficile d'effectuer une bonne interprétation des informations géographiques connues pour les transformer, en respectant leur inter-relation, en données dessinables sur le support choisi dans le cadre d'objectifs particuliers des utilisateurs, en suivant l'ensemble des opérations subtiles effectuées actuellement par l'homme. Réciproquement, avec un vaste ensemble d'informations et d'opérations potentiellement accessibles, il est délicat de déterminer quelles informations sont les plus pertinentes pour effectuer telle opération plutôt que telle autre pour résoudre un problème précis. La communauté cartographique est

[1] Comité Français de Cartographie
[2] Association Cartographique Internationale

convaincue que la GC ne trouvera sa juste place parmi les outils de communication et d'aide à la décision que lorsqu'elle aura réussi sa mutation informatique. On se pose donc des questions:

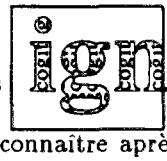
- *quels sont les problèmes de la GC?*
- *comment ces problèmes sont-ils représentés?*
- *quelles sont les techniques informatiques disponibles face à un tel problème?*

C'est autour de ces aspects que s'organise cette thèse. Bien entendu un certain nombre de préalables seront indispensables afin de bien préciser le problème de la GC. Une fois défini un cadre conceptuel nous étudierons les principales idées et techniques de l'intelligence artificielle. Après avoir donné quelques aspects de la modélisation des opérations de la GC et leur formulation mathématique et graphique, nous décrivons une méthodologie de représentation de connaissances qui, pourvue d'une interprétation adéquate et basée sur le concept de langage orienté objet et de règle de production, permet de coder les connaissances de la GC pour traiter les problèmes complexes. Enfin nous donnerons quelques résultats basés sur les détections et déductions par le moteur d'inférence écrit en Le-lisp en lui fournissant des connaissances initiales à l'aide des exemples, et des illustrations graphiques sorties par des codes PostScript.

1.1 Aperçu de la généralisation cartographique

Comparez cette  originale et cette  réduite de la précédente : dans cette dernière, il n'y a pas assez de place pour contenir le mot **carte**. Un traitement spécial est donc presque obligatoire si l'on veut garder le caractère exploitable de cette carte.

Une diminution d'échelle provoque une diminution de surface disponible alors que l'encombrement de certaines symbolisations (ex: les écritures, les signes conventionnels des routes, etc.) sur une carte reste par contre plus ou moins constant.

La première méthode est de réduire la taille de **carte**:  ou bien:  avec une taille encore plus petite. Mais on ne peut pas réduire à l'infini les tailles des symboles quand l'échelle devient de plus en plus petite, l'acuité visuelle de l'homme lui interdit la connaissance des objets infiniment petits sans l'aide d'appareils optiques. Par exemple, on voit bien les lettres **cogit** dans  , tandis que dans  , il devient beaucoup plus difficile de les reconnaître après une simple réduction de 50% en bloc. Dans ce cas, une resymbolisation est nécessaire.

Une carte représente des objets géographiques que malheureusement nous n'avons pas matériellement la possibilité de voir en totalité dans la nature et dont, au contraire, nous ne percevons que des détails à notre échelle humaine, avec toutes les déformations et les impressions qu'entraîne notre situation normale au ras du sol.

Une carte bien conçue et bien réalisée permet à la fois d'embrasser d'un seul coup d'oeil un objet géographique d'une surface quelconque et d'en distinguer les éléments significatifs quelles qu'en soient leurs dimensions réelles. Le choix de ces éléments, la simplification de leurs représentations, leur harmonisation, leur mise en oeuvre coordonnée définissent la *généralisation*.

Chaque carte représente une réduction de l'espace, alors que les graphismes, pour qu'ils soient lisibles, occupent une place proportionnellement égale ou plus grande que dans la réalité. Le problème est donc le suivant: compte tenu de l'échelle et de la nature des graphismes, quels sont les objets qui doivent être conservés et quels sont ceux qui doivent être éliminés? Comment symboliser ceux qui sont retenus en fonction de la surface du support?

La généralisation est aussi nécessaire:

- à la rédaction d'une carte en changeant le but d'une carte initiale,
- à la dérivation d'une carte en diminuant l'échelle d'une carte d'échelle plus grande.

Donc en résumé, la GC est nécessaire pour résoudre les difficultés à concilier la clarté de la carte avec les contraintes dues à l'échelle et aux objectifs particuliers des utilisateurs.

Le but de la généralisation cartographique est d'aboutir à une synthèse plus poussée de l'information et de résoudre le problème d'encombrement à l'aide des 3 opérations:

sélection, schématisation et harmonisation

Elle permet finalement de reconstituer sur une carte la réalité de la surface représentée dans ses traits essentiels en fonction du but de la carte, de son thème, de son échelle et des particularités de la région cartographiée.

La généralisation cartographique est une sorte de caricature, une sorte d'art, elle déforme presque obligatoirement les formes des données par la réduction, resymbolisation, etc. Un problème important est d'assurer la qualité du produit pendant tout le processus. C'est-à-dire qu'il faut préserver la fiabilité des données, la lisibilité de lecture (communication effective du produit cartographique), la précision compatible avec l'échelle et l'esthétique de la carte.

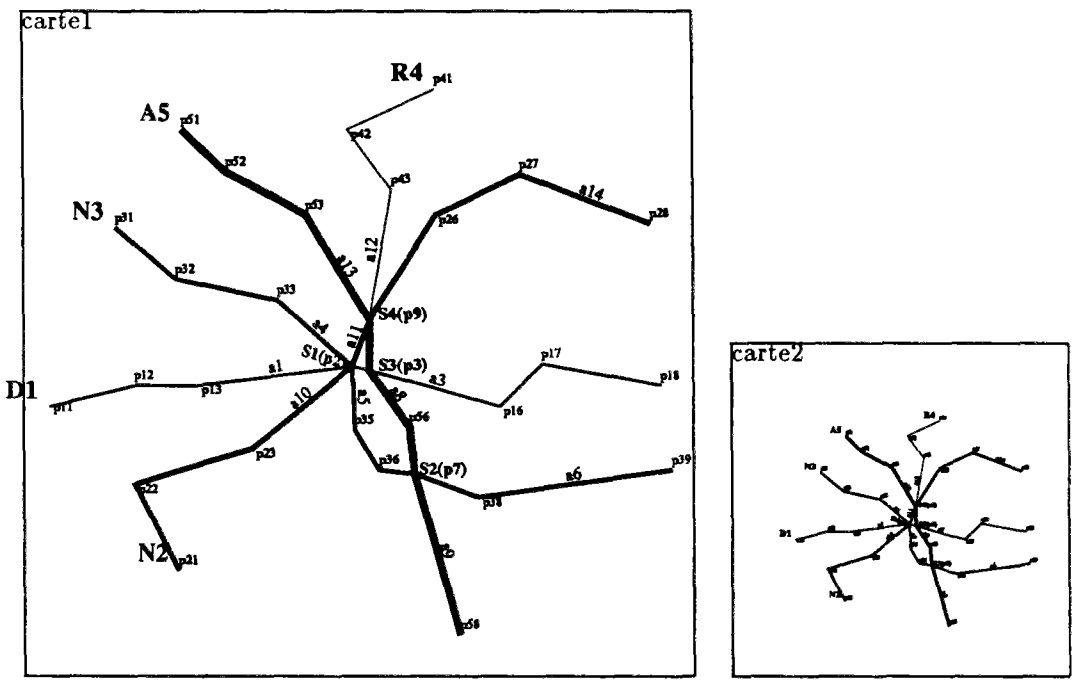
Un compromis entre le désir d'exactitude et de clarté devrait être assuré; cette expertise, véritable casse-tête de la généralisation cartographique, est la plus difficile à formuler.

La figure 1.1 montre un exemple simple.

1.2 Difficultés essentielles

Le principe général de la généralisation cartographique (G.C.) repose sur les processus complexes et interdépendants de sélection, de schématisation et d'harmonisation des éléments constitutifs de la carte, en fonction des facteurs suivants:

- but et thème de la représentation,



carte1: les carrefours au S1 et au S3 sont à la limite de distance de séparation.

carte2: après la réduction, on ne distingue plus le carrefour au S1 de celui au S3; les labels ne sont plus aussi lisibles.

Figure 1.1: Un simple exemple d'un carrefour à généraliser

- échelle: elle a une influence primordiale sur la généralisation.
- caractère géographique régional: selon que l'on est en rase campagne ou en zone urbaine concentrée, la précision sera très différente.
- niveau de lecture,
- règles de lisibilité,
- valeur et précision des données.

Plus de détails se trouvent dans le chapitre 2 et aussi dans les références suivantes: [1], [7], [8].

Depuis longtemps, la G.C. est un des aspects les plus délicats de la cartographie à cause des difficultés suivantes:

- Le volume des données est important, les structures de données sont complexes.

- La procédure subjective des opérations n'est pas homogène. Le résultat de la simplification d'une ligne par le même cartographe à quelques jours d'intervalle pourrait être même significativement différent. Une carte à échelle deux fois plus petite ne peut pas être considérée comme étant l'homologue de sa grande soeur avec seulement quatre fois moins d'objets représentés. Les yeux humains n'arrivent plus à distinguer les choses qui sont en-dessous du seuil de $0.1mm^2$.

- Les jugements humains sur la qualité de produit sont difficiles à formuler, vu l'absence de critères universellement acceptables.

  Des considérations de nature esthétique et des critères de valeur intrinsèques qui méritent d'être définis interviennent dans le choix d'une appréciation personnelle souvent subjective. Devant une collection de cartes représentant le même phénomène, les avis peuvent être aussi différents.

  Le seul critère constituant une référence universelle est peut-être celui de l'*efficacité.*

- La cartographie est une sorte d'art qui fait appel au bon sens et à l'expérience, mais il est difficile de formaliser le savoir des experts, de même qu'il est pratiquement impossible de formuler mathématiquement tous les savoirs humains.

En bref, la généralisation ne peut pas être codifiée ou réglementée de façon précise. Cette constatation laisse le champ libre à l'empirisme et aux interprétations individuelles du cartographe, préparateur ou dessinateur; elle conduit donc inévitablement à des productions personnalisées dont le contenu et la forme dépendent plus de leurs auteurs que de règles rationnelles et méthodiques, ce qui justifie l'importance de l'harmonisation.

Les moyens actuels de recueil, de traitement et de transcription graphique de données ne seront efficaces que dans la mesure où les fondements scientifiques de la généralisation auront été élaborés, en prenant les meilleurs outils informatiques pour la réalisation. C'est une tâche difficile et délicate car la cartographie est tributaire de son passé et des habitudes acquises. Il semble que c'est un champ privilégié d'application des recherches en intelligence artificielle: la généralisation cartographique semble indissociable des techniques de l'IA.

1.3 Eléments d'intelligence artificielle

L'intelligence artificielle (IA) est depuis quelques années un point de focalisation des média, du public et des scientifiques. On s'interroge sur son existence en tant que discipline scientifique, sur ses buts et son champ d'investigation, sur sa situation et les limites qu'elle pourrait atteindre, sur ses applications actuelles ou à venir et leur impact social, économique ou psychologique[12]. Aujourd'hui, elle suscite un grand engouement dans l'industrie.

Le concept d'*intelligence* est complexe et relatif, et l'expression fortement controversée, d'*Intelligence Artificielle* ne le précise pas davantage.

Bien que les Chinois doutent du caractère inné de l'intelligence humaine, ils la nomme l'*intelligence du ciel.* La philosophie du donné, empirisme, considère que toute connaissance vient de l'expérience[17], c'est à dire que l'esprit est une pure passivité,

une simple faculté d'enregistrer des réalités indépendantes de lui. L'empirisme est un procédé spontané qui consiste à agir par tâtonnement et sans réflexion préalable.

Toute notre connaissance commence avec l'expérience, disait KANT, *mais il n'en résulte pas qu'elle dérive toute de l'expérience.* Il semble, en effet, qu'il y ait des vérités indépendantes de toute expérience particulière, notamment en logique et mathématiques.

Les théories de Wallon et de Piaget donnent du développement intellectuel l'image d'une hiérarchie de savoir-faire acquis par l'enfant à travers une série de *stades*. Au plus bas niveau se situent les comportements réflexes, puis viennent les comportements concrets, liés à une *intelligence pratique* qui existe déjà chez les animaux supérieurs; l'*intelligence concrète* est le premier niveau de l'*intelligence discursive* capable d'opérer sur des signes et des symboles; quand cette opération se libère de l'affectivité et de l'action immédiate, l'enfant atteint le stade de l'*intelligence abstraite* (capacité de distinguer les qualités d'un objet, de comparer, de classer), que complète l'*intelligence conceptuelle* (raisonnement hypothético-déductif, appel à une logique formelle).

Les humains pensent qu'ils sont des êtres plus intelligents que les animaux; en étant fier, ils cherchent toujours à reproduire les intelligences humaines, ce qui, avec les progrès des ordinateurs a fait naître l'*intelligence artificielle*. Son principal objectif est d'obtenir soit un *résultat* voisin de ce qu'aurait obtenu l'homme avec éventuellement des démarches différentes, soit d'un ordinateur un *comportement* semblable à celui d'un humain, c'est à dire, de disposer d'un programme qui adopte une démarche analogue à celle d'un spécialiste confronté à un problème précis. Dans cette dernière optique, le travail des chercheurs en IA commence par des observations de comportements humains et animaux faces à des situations particulières. De ces observations, on cherche à constituer un ensemble de théories scientifiques cohérentes, de lois de fonctionnement du raisonnement. Finalement, cette analyse longue et difficile doit aboutir à la reproduction, par des moyens réellement artificiels (mécaniques, informatiques), des comportements étudiés.

L'ensemble des théories et des techniques mises en oeuvre pour réaliser des telles machines constitue l'*intelligence artificielle*.

L'informatique *classique* traite des problèmes de manipulation de données: essentiellement des valeurs numériques ou des chaines de caractères. Un programme y est chargé de traiter de manière spécifique un type de données bien propre.

Les logiciels d'IA, par contre, doivent traiter non seulement des nombres et des caractères mais des informations plus abstraites: des connaissances humaines au milieu des activités concrètes, jusqu'ici faiblement informatisées, du type diagnostic, prévision, conception, planification d'actions; la généralisation cartographique en est un bon exemple.

De telles activités intellectuelles sont souvent, comme nous l'avons constaté précédemment en ce qui concerne la généralisation cartographique, difficiles à être représentées sous forme d'algorithmes sûrs et définitifs.

Comme toute discipline scientifique, le but de l'IA est de comprendre les phénomènes et les processus qui interviennent dans son champ d'investigation qui est le *raisonnement*:

- *l'IA cherche à comprendre les mécanismes de réflexion*, son outil central d'investigation est l'*ordinateur*, elle cherche à marier les deux démarches *cognitif* et *informatique numérique*.

Comment rendre les ordinateurs plus habiles? telle est la question fondamentale de l'IA.

- l'IA cherche à concevoir des programmes et des machines en mesure de traiter des *problèmes* pour lesquels on ne connaît pas de méthodes de résolution directes et assurées.

  Elle comprend tous les domaines où nous traitons de l'information sans savoir comment nous nous y prenons. Ces domaines sont nombreux et recouvrent les diverses formes de raisonnement, de résolution de problèmes tels que la généralisation cartographique.

- L'IA interroge et cherche à découvrir le réel, les mécanismes de raisonnement et de compréhension.

  L'IA apporte sa contribution à la compréhension des manipulations effectuées sur des objets au cours de l'exercice de facultés mentales telles que: interpréter, comparer et faire des associations, synthétiser, abstraire et mémoriser, déduire, généraliser, induire, apprendre ....

- L'IA exploite les possibilités du réel pour concevoir des machines et des programmes traitant une classe de problèmes particuliers tels que: analyser une image, identifier son contenu, faire une synthèse plus poussée de son contenu pour générer une carte à échelle plus petite.

  Pour appréhender ces problèmes, les chercheurs s'intéressent aux techniques de représentation sur ordinateur des connaissances symboliques, aux règles et algorithmes d'inférence logique, aux machines, et aux logiciels les plus adaptées à ces représentations et ces algorithmes.

L'objectif visé est que *ça marche*: que le programme ou la machine conçus résolvent effectivement le problème posé intelligemment comme un expert humain, mais comment rendre une machine aussi intelligente qu'un spécialiste humain confronté à un problème précis?

En fait, une activité humaine intelligente met en oeuvre une grande quantité de connaissances, comprenant des informations générales mais aussi des informations propres au champ de l'activité, ainsi que des stratégies de raisonnement permettant de manipuler ces informations.

Entre l'*intelligence du ciel* et *les connaissances pratiques de l'empirisme*, on peut se mettre d'accord sur le fait qu'un humain développe son intelligence, et renforce son intelligence en accumulant une masse de connaissances héritées des ancêtres au cours des siècles, et devient ainsi de plus en plus mûr ou intelligent, et finalement, expert. Un expert n'est pas nécessairement intelligent: il sait, sans réfléchir. *Beaucoup de ce qui apparemment nécessite de l'intelligence peut être réalisé par compilation des règles de comportement superficiel*[12].

On s'oriente vers la voie: mettre à la disposition des programmes une masse importante de connaissances (déductives) sur le sujet traité, et par là même, les spécialiser dans un secteur bien précis, c'est à dire qu'on commence par apprendre aux ordinateurs, on les forme comme on forme des humains, ensuite, on espère qu'ils travailleront comme (plus ou moins) un humain, mais toujours sous la direction des êtres humains, au moins ceux qui écrivent le programme et ceux qui l'utilisent. La connaissance repose donc sur trois composantes: des *faits* ou données brutes qui

caractérisent les objets du domaine considéré, des *règles* permettant de manipuler ces faits, des *stratégies* de raisonnement et des heuristiques exprimant la façon de se servir des règles des structures de données appropriées au stockage et à la manipulation des informations relatives à une application.

D'importants travaux sont déjà faits sur ce domaine. De tels programmes existent, leur but est de parvenir, dans des domaines bien précis, à des performances avoisinant celles des meilleurs spécialistes. On les nomme *systèmes experts* (SE), ou *systèmes à base de connaissances*.

Les techniques de base de l'IA reposent essentiellement sur deux concepts permettant de représenter de manière efficace une grande variété de connaissances. Le premier, les règles de production et la programmation logique, est un outil popularisé par les systèmes experts (chapitre 3.1). Le second moyen, également très utilisé, est la représentation sous forme d'**objets** (chapitre 4).

1.4 Perspectives

Les deux sciences, la généralisation cartographique et l'IA, ne sont pas des sciences aussi mûres et rigoureuses que les mathématiques. Néanmoins, on espère les marier ensemble par une étude des techniques existantes en informatique autant qu'en cartographie.

La vocation d'un système expert est d'aider à résoudre des problèmes très difficiles pour l'homme relevant d'un domaine de compétence particulier où les activités sont difficilement représentables sous forme d'algorithmes sûrs et définitifs comme on l'a déjà indiqué plusieurs fois dans le domaine de la généralisation cartographique.

On espère contourner cette difficulté partiellement en faisant appel à la méthodologie des systèmes-experts: leur structure et leur fonctionnement se prête assez bien à la reproduction des facultés humaines de décision ou jugement, à partir non d'algorithmes complets mais d'éléments d'algorithmes, spécifiques d'un domaine d'application, plus ou moins épars et non définitifs, fournis par des experts humains du domaine considéré.

Par une étude critique des cartes existantes et par une analyse objective des facteurs de généralisation, quelques spécialistes ont déjà montré que certains principes, et notamment ceux de la sélection, peuvent recevoir une formulation mathématique et que les directives générales résultant de l'expérience et de la tradition peuvent s'exprimer sous une forme cartésienne.

La structure complexe de l'image cartographique est régie par une multitude de règles qui sont appliquées plus ou moins implicitement par le dessinateur, selon son expérience et son habileté; aussi, sans nier l'importance de ces facteurs humains, il importe de limiter leur influence pour réduire la part d'interprétation personnelle, codifier ce qui peut l'être et normaliser ainsi les résultats et la qualité de la généralisation.

Notre but est:

- de constituer une approche du problème par une analyse objective des facteurs de généralisation, en recueillant des règles qui sont appliquées plus ou moins implicitement par le cartographe, tout en essayant d'en tirer des règles générales, de formuler et codifier ce qui peut l'être et normaliser ainsi les résultats et la qualité de la généralisation.

- de montrer, dans quelques cas bien précis, comment il est possible de tendre vers cette normalisation.
- de concevoir un système capable de réaliser les opérations de la G.C. ayant recours aux techniques diverses de l'intelligence artificielle, en cherchant une certaine automatisation et la souplesse des processus, apportant la rapidité et l'économie, et peut être une qualité supérieure et une plus grande homogénéité du travail.

Le système a les fonctionnalités suivantes:

- Saisir et modifier des connaissances.
- Rédiger des cartes d'une échelle donnée à partir des données d'une échelle plus grande.
- Résoudre le problème qu'on lui demande dans le cadre de ses capacités, et dialoguer avec l'opérateur humain.
- Justifier la trace de sa résolution.

Le système comprend les modules suivants (voir la figure 1.2):

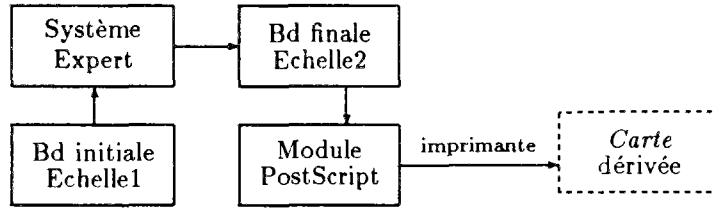


Figure 1.2: Système essentiel de G.C.

- une base de connaissances contenant la base des données des objets initiaux et la base d'expertise (la base des règles).
- un moteur d'inférences non monotone, avec variables, fonctionnant en chaînage avant, avec l'interrogation de la base de faits, etc.
- un module pour acquérir des connaissances, les visualiser, et les compiler sous forme interne.
- un module de traduction des données graphiques en PostScript.

Partie I

Cartographie

Chapitre 2

La généralisation cartographique

Ce chapitre décrit les différents aspects du processus de la GC pour bien comprendre et connaître ses problèmes.

2.1 Analyse du problème

Une carte est plus qu'une simple image visuelle ou photographique d'une région donnée.

- Elle assure, par l'utilisation des deux dimensions du plan, un classement géographique et, par le jeu des variables visuelles (voir le tableau 3.1 de la page 46), une perception de la nature et des valeurs des phénomènes.
- Elle n'est pas un enregistrement pur et simple, elle communique à son lecteur une information traitée et ordonnée compte tenu de l'échelle, d'un espace géographique défini.
- Elle constitue le moyen le plus efficace pour enregistrer, calculer, révéler, analyser et comprendre les relations spatiales qui existent entre les différents phénomènes concrets ou abstraits dont la localisation est géographique[1].

Outre l'aspect scientifique que la définition[1] de l'ONU a attribué à la cartographie, la communauté cartographique parle de l'ensemble des études et opérations *scientifiques*, *artistiques* et *techniques*.

A l'inverse d'une photographie qui enregistre fidèlement tous les détails compatibles et donne des détails une image qui correspond à leur dimensions originales (voir sur la figure 2.3 les réductions à la façon photocopieuse d'une carte), une carte opère une sélection, elle ne retient et renforce que les détails jugés indispensables en fonction des besoins et de l'échelle de la carte et elle omet volontairement tous les autres. Une carte réalise un classement selon des critères qualitatifs ou quantitatifs bien définis en

[1] La Cartographie est la science qui traite de l'établissement de cartes de toutes sortes. Elle englobe toutes les phases des travaux depuis les premiers levés jusqu'à l'impression finale des cartes.

parole:	*carte*:
▸ prononcer les phonèmes les uns à la suite des autres,	▸ enregistrer des données élémentaires,
▸ grouper les phonèmes en mots et en phrases,	▸ traiter ces données, les mettre en ordre et les coordonner,
▸ exprimer des idées au moyen d'une ou plusieurs phrases complètes.	▸ passer un message et des idées au moyen des graphiques: carte.

Tableau 2.1: Comparer une *carte* avec une *parole*

fonctions de ses buts et elle souligne les détails les plus importants selon ces critères. Par exemple, les routes sont classées par leur largeur, le nombre de voies, l'importance du trafic, et elles sont représentées par des figurations symboliques et conventionnelles comme les signes conventionnels en fonction d'échelle. La figure 2.4 montre des réductions de la carte de Marmande en prenant les signes conventionnels correspondants des routes.

Une carte bien conçue fournit immédiatement une image claire et lisible qui procure des informations à tous leurs utilisateurs quel que soient leur langue maternelle et leur degré de connaissances.

La cartographie, un moyen de communication et d'expression, est un langage graphique à part entière, qui possède comme tous les autres langages formels:

son vocabulaire : elle utilise un ensemble des symboles graphiques qui suggèrent des objets ou des idées individualisées,

- ▸ une ligne de couleur rouge représente une autoroute;
- ▸ un carré désigne une maison, etc.

sa grammaire : c'est la façon d'agencer les symboles graphiques, les combinaisons et les groupements exprimant des idées d'un ordre supérieur par comparaison, association ou analogie avec d'autres concepts.

- ▸ la ligne d'une route ne peut pas traverser une maison sans tunnel;
- ▸ une voie ferrée ne doit pas croiser à niveau une autoroute;

ses règles : règles de fiabilité de transfert des informations, règles de lisibilité des données, règles esthétiques découlant de la culture:

- ▸ les informations dans une carte sont traitées pour dégager et traduire, sous une forme aussi facile que possible à lire et à retenir, les lois de correspondance et les relations qui existent entre ces informations.
- ▸ une carte ne doit pas donner de fausses informations telles que Paris soit en dehors de la frontière française;
- ▸ Le fait que la couleur rouge soit attribuée à une forêt est susceptible de troubler lecteur moyen.

Par rapport aux autres formes de langage, la représentation cartographique dispose d'ailleurs d'avantages importants:

- Par nature, la mémoire visuelle symbolique est plus efficace que la mémoire phonétique abstraite, comme le confirme l'histoire: les écritures symboliques sont beaucoup plus anciennes que les écritures alphabétiques. L'homme assimile naturellement les dessins.

- L'utilisation des deux dimensions du plan et la grande universalité des variables visuelles offrent de grandes possibilités d'expression. Un petit dessin sera plus précieux pour un étranger qui cherche son chemin et ne connaît pas bien la langue du pays qu'une petite phrase confuse.

Devant une collection de cartes représentant le même phénomène, les appréciations personnelles peuvent différer tant sur l'esthétique que sur la nature de la carte. Le critère de l'*efficacité* est considéré comme une référence universelle.

2.2 Cas des voies de communication

Vue la complexité totale de la généralisation cartographique, on a préféré, en première étape, prendre un cas particulier comme domaine de travail: le cas de voies de communication en espérant en faire un modèle de base pour proposer une méthodologie de représentation des connaissances dans ce domaine.

2.2.1 Carrefour

Un point est géométriquement d'aire nulle. Comme toute représentation, pour être visible, la représentation d'un point est nécessairement une surface d'aire non nulle; et cette surface est purement conventionnelle et seul son centre a une signification positionnelle.

Ponctuel comme première caractéristique, un carrefour peut être défini par l'intersection de deux lignes matérialisées, son sommet, parfaitement défini par l'intersection d'axes, sans surface; mais un carrefour en ce qui nous concerne, a une surface qui correspondent aux largeurs des routes qui s'y rejoignent.

2.2.2 Voies

La ligne est un lieu du plan de mesure nulle dans un espace à deux dimensions, et non nulle (longueur) dans un espace à une dimension.

Une voie de communication peut être représentée:

- par une ligne d'épaisseur de 2 mm;

- par une ligne continue ou discontinue;

- par des tracés jumelés.

mais, seul l'axe linéaire de la tache visible a une signification positionnelle.

Donc, une voie peut varier en largeur, en valeur, couleur .... On fera appel à la largeur, à la variation et à la couleur de la ligne pour faire apparaître une hiérarchie.

Par exemple un cas sur la figure 1.1, étant donné: un ensemble des carrefours $\Sigma = \{e_1, e_2, e_3, \ldots, e_n\}$ dans un réseau routier: **S1, S2, S3**; On doit

- chercher à les représenter sous un modèle cartographique, mathématique, physique, informatique, etc.
- trouver des relations **R** entre les Σ; prenons l'exemple entre le carrefour **S1** (p2) et **S3** (p3) de la figure 1.1:
  - **S1** (p2) et **S3** (p3) ont un arc(a2) en commun
  - **S1** (p2) est à la gauche de **S3** (p3);
  - **S1** (p2) et **S3** (p3) sont à la limite de lisibilité;
  - etc.
- prendre en compte les contraintes **C** suivantes:
  - échelle,
  - but,
  - caractère des données,
  - lisibilité.
  - esthétique.

Les règles de lisibilité sont numérisables entre les éléments bien définies et restent constantes une fois modélisées. Alors que les échelles, buts et caractères des données sont variables et posent des problèmes pour la modélisation de l'ensemble des opérations de la généralisation.

2.3 Les facteurs d'influence

Le principe général de la généralisation cartographique repose sur les processus complexes et interdépendants de sélection, de schématisation et d'harmonisation des éléments constitutifs de la carte; les facteurs d'influences sont les suivants:

- *but* et *thème* de la représentation. L'influence du but de la carte est évidente: si on compare 2 cartes géographiques générales à la même échelle, l'une dans un atlas de référence, l'autre dans un atlas scolaire, la précision sera complètement différente. L'influence du thème concerne l'harmonisation et l'équilibre des éléments représentés. La carte topographique est nécessairement détaillée puisqu'elle constitue le cadre dans lequel s'intégreront toutes les autres productions. Mais on ne peut pas satisfaire intégralement tous les besoins, car une carte très complète mais parfaitement illisible est inefficace pour des non spécialistes.
- *échelle*. Elle a une influence primordiale sur la généralisation. Théoriquement, une carte à 1:50 000 aura beaucoup moins de détails que sa grande soeur à 1: 25 000, le nombre de phénomènes représentables dans une surface donnée varie

en fonction inverse du carrée du rapport d'échelle. Mais une carte de 1:100 000 n'aura pas forcément 4 fois moins d'élément que sa petite soeur de 1:50 000, car par symbolisation, on arrive à mettre plus d'objets que le nombre théorique. Dans les cartes à grandes échelles, le souci est de maintenir la précision graphique absolue, ce qui conduit à une densité de détails inférieure à celle qu'exigerait le seul critère de lisibilité, Aux petites et très petites échelles, on se contentera d'avoir la précision relative, donc le nombre d'éléments maintenus est souvent supérieur à celui qu'autorise le rapport des surfaces.

Avec la réduction d'échelle, la généralisation n'est pas seulement obligatoire, elle est nécessaire et dans l'esprit de la cartographie: une autoroute qui constitue un phénomène essentiel à l'échelon national ou international, n'a pas beaucoup plus de valeur dans le cadre de la commune qu'elle traverse que le moindre chemin d'exploitation.

- *caractère* géographique régional. La carte d'une région a pour but de traduire avec le maximum d'efficacité le caractère et les traits essentiels de cette région et de fournir des points de comparaison entre des régions semblables. Les détails des sentiers, points d'eau seront plus ignorés dans une région tempéré de fort peuplement que dans une zone désertique.

- *niveau de lecture.* Comme la transcription graphique est totalement dépendante des lois naturelles de la vision qui lui imposent des limites et des règles strictes, la carte ne peut que répondre à un objectif bien précis. Elle doit être conçue en fonction du niveau de lecture pour lequel la lecture sera instantanée et cela explique les réactions diverses devant une collection de cartes.

- *règles de lisibilité.* Ce facteur est entièrement développé dans la section suivante, vu sa grande importance.

- *valeur et précision* des données de base. Evidemment ce facteur secondaire est également important, la nature des documents disponibles conduit en effet le cartographe à prendre des décisions différentes.

Dans tous les cas, quels que soient le niveau de lecture et la fonction de la représentation, l'efficacité d'une image est maximale lorsque le temps de perception est le plus court[1].

2.4 Règles de lisibilité

Normalement, les cartes sont destinées à être examinées sans l'aide d'instruments grossissants, à une distance normale (de l'ordre de 30cm) et sous un éclairage moyen. Dans ces conditions, leur contenu doit être conçu de telle sorte que tout élément graphique isolé soit perceptible, que sa forme soit distinguée, que deux éléments voisins soient séparés, enfin que les différents paliers soient différenciés.

Voir le tableau 2.2 de la page 26.

- Un point est perceptible lorsqu'il est vu sous un angle de 1'; le seuil de perception pour un élément ponctuel est alors de 0,09mm, diamètre de ce point, ou dimension minimale d'un élément graphique de forme irrégulière.

Contraintes visuelles		Représentations graphiques						
Acuité visuelle de discrimination		à 30cm de distance, dd doit être de $dd \sphericalangle = 1'$						0.09 mm ≈ 0.1 mm
Acuité visuelle d'alignement		da doit être au moins de 0.02 mm						
Seuil de perception	ponctuel ⊸ mm	● 0.2	○ 0.3	■ 0.4	2 0.6	△ 1.0	⅄ 0.5	⌔ ϕ0.6
	linéaire ⊸ mm	0.08 en tracée			0.1 en dessin			
Seuil de séparation	⊸ mm	0.2		0.3 0.3		0.2		
Seuil de différenciation	ponctuel ⊸	entre 2 paliers, le rapport des surfaces doit être au moins de 2						
	linéaire ⊸			rapprochés		écart d' épaisseur		0.1 mm
				éloignés				0.3 mm

Tableau 2.2: Tableau de dimensions graphiques minimales: Les chiffres sont exprimés en **mm**

- Une ligne, du fait de sa continuité, admet des tolérances plus faibles, soit un angle visuel de 0.7' ou une épaisseur de 0.6mm.
- Dans le domaine des formes, en implantation ponctuelle ou linéaire, il faut tenir compte de la lisibilité angulaire qui est faible lorsque les angles se rapprochent de 0^0 ou de 180^0 et maximale aux environs de l'angle droit;
- Un carré de petite dimension se distingue plus aisément d'un cercle de même surface qu'un hexagone ou qu'un pentagone. Selon l'expérience, un carré poché est distingué avec une largeur de 0.4mm, un carré vidé sera visible avec une largeur de 0.6mm.
- Les sinuosités d'une ligne anguleuse de .1mm se distinguent lorsque leur coté est d'environ 0.6mm-0.7mm, si elles sont quelconques, et leur base est de l'ordre de 0.6mm-0.7mm et leur amplitude de 0.4mm.
- Pour séparer deux signes voisins, on définit un seuil de séparation, qui est l'écart minimal nécessaire entre deux éléments graphiques voisins pour pouvoir les isoler à l'oeil. Par exemple, pour deux lignes fines parallèles l'écartement angulaire est un espacement graphique de 0.17 **mm**, et pour des lignes épaisses 0.15 **mm**. C'est cette valeur qui constitue le seuil de séparation valable pour des signes pochés contigus.

En fait, dans la pratique, les chiffres exprimés ci-dessus sont nettement plus élevés; car il ne suffit pas de distinguer ou de séparer les éléments graphiques, il faut encore apprécier les différences de dimension ou de valeur qui expriment des paliers distincts en classement quantitatif (ou simplement ordonné), et corrélativement les éléments identiques appartenant au même palier. Les écarts sensibles qui définissent le seuil de *différenciation* doivent être respectés si la lecture au stade élémentaire est indispensable. Si au contraire, les questions élémentaires sont inutiles et qu'on ne souhaite pas une perception sélective des détails mais une vision d'ensemble, les écarts seront plus faibles et des tailles extrêmes identiques.

En cartographie topographique, les différenciations sont réalisées par des combinaisons des diverses variables visuelles qui évitent, dans une certaine mesure, l'utilisation de signes occupant trop de place.

En bref, pour qu'une carte soit parfaitement lisible, il est indispensable de rechercher un équilibre, d'une part entre la surface totale des différents éléments graphiques et la surface du support (densité graphique), d'autre part dans les contrastes des graphismes entre eux.

2.5 Problème des signes conventionnels

2.5.1 Nécessité des signes conventionnels

Une des caractéristiques essentielles de la représentation de la planimétrie est qu'autant que possible, les détails doivent apparaître suivant la projection de leur contour, réduits à l'échelle, ou à la rigueur, par des graphismes axés ou centrés sur l'emplacement de cette projection.

Par les règles de lisibilité, on a défini les dimensions minimales des images cartographiques. Donc théoriquement, à l'échelle de 1/25 000, on pourra conserver la représentation en projection par un trait isolé de tous les objets linéaires mesurant au moins 2.5m de largeur (0.1mm sur la carte) et par deux traits séparés de tous les objets linéaires mesurant au moins 10m de largeur (traits de 0.1mm séparés par un espace de 0.2mm). De même, les détails à implantation zonale seront perceptibles dès que leur coté dépassent 12.5m. Toutefois ces minima absolus ne permettent aucune différenciation d'objets de nature différente mais de dimension identique. Cette différenciation, justifiée et nécessitée par la multiplicité des objets, demande, dans tous les cas, une amplification des dimensions théoriques.

En fait une représentation parfaitement homométrique, c'est-à-dire une représentation en projection et dimension réelles des objets, n'est possible qu'à très grande échelle, jusqu'à 1/5 000. Les cartes topographiques font donc tout naturellement appel aux signes conventionnels souvent pour les éléments ponctuels et linéaires, puisque la variable *forme* (tableau 3.1) est la seule qui s'adapte au grand nombre d'objets.

Pour des besoins de commodité, on a l'habitude de répertorier les éléments du contenu planimétrique d'une carte selon un certain nombre de catégorie. Par exemple, la catégorie linéaire comprend: les routes, chemins, sentiers, voies ferrées, les éléments du réseau hydrographique, les limites de culture et les limites administratives, etc.; alors que les constructions diverses sont dans la catégorie ponctuelle; la végétation dans la catégorie zonale.

Dans la catégorie linéaire qui constitue notre domaine de travail, on se limitera aux: **voies de communication** du **réseau de communication**.

2.5.2 Les signes conventionnels linéaires

Pour les implantations linéaires, la forme générale est imposée par le tracé, et une figuration semblable à la projection horizontale s'impose. Les critères d'ordre des distinctions de détails se traduisent par des variations de largeur du signe, par des différences de grosseur de traits ou par des variations de valeur (grisé ou couleur, discontinuités des traits). Ces diverses possibilités favorisent les combinaisons; c'est ainsi que, pour les routes, les critères de nature du revêtement, de classement administratif

Objets		*Échelle*					
		1:25 000	1:50 000	1:100 000	1:250 000	1:500 000	1:1M
Voies ferrés	1 voie	30					
	+ 2 voies	20.30.20 (70)	30	40	30		20
Autoroute		14.41.8.41.14 (118)		25.41.8.41.25 (140)	20.30.12.30.20 (112)	12.75.12 (99)	20.30.20 (70)
R	2x2v chaussées séparés			8.40.8.40.8 (104)			
O	+3 voies	14.70.14 (98)		12.75.12 (99)	8.60.8 (76)	10.35.10 (55)	40
U	2 voies larges	12.50.12 (74)					
T	2 voies étroites	10.35.10 (55)				30	
E	moins de 5m	couleur orange tireté blanche		12.40.12 (64)	8.30.8 (46)	25 tireté	20
Autre route non classée	étroite						
	irrégulière	.....			20 continu	20 tireté	0 0
Chemin d'exploitation		20 continu			20 tireté		0
Sentiers		20 tireté					0 0
Rivière	navigable	vrai grandeur			signe conventionnel		
Villes	+5000h	zonal					⊙
	+3000h	zonal				2 160	⊙
	- 3000h	zonal			⊙130	⊙130	⊙

Tableau 2.3: Tableau de l'évolution des signes conventionnels avec l'échelle : Les chiffres sont exprimés en 1/100 mm; dans 20.30.20 (70): 20: un trait noir ou de couleur de 20/100mm, 30: espace entre deux traits, 70 est le largeur total du signe conventionnel

et de largeur de voie se superposent éventuellement. Si le classement est quantitatif, on fait généralement varier la largeur du signe mais, pour éviter un encombrement excessif et une amplification disproportionnée, on préfère, aux petites échelles, jouer sur le nombre de signes adventifs comme des traits pointillés.

Le tableau 2.3 montre l'encombrement des différents signes conventionnels suivant les différentes gammes d'échelles.

2.5.3 Evolution des signes avec l'échelle

Il est pratiquement impossible de conserver les mêmes signes conventionnels pour toute la gamme des échelles. Comme les signes conventionnels, pour être lisibles, amplifient de plus en plus leur emprise graphique au fur et à mesure que l'échelle diminue, ces normes de sélection sont absolument nécessaires à chaque stade de dérivation qui conduit de la carte de base aux cartes générales mais il est prudent de considérer le problème dans son ensemble. En effet, la part de convention croît en même temps que diminue le nombre de phénomènes qui peuvent être représentés par la projection de leurs contours. Simultanément l'espace disponible varie en fonction inverse du carré des rapports d'échelle alors que les dimensions minimales des graphismes disponibles restent invariables et que les dimensions réelles des objets constituent un ensemble très vaste.

Ces données impliquent, lorsqu'on descend la gamme des échelles, d'une part une généralisation par sélection, harmonisation et schématisation, d'autre part une égalisation de l'importance apparente des détails conservés; on peut dire que progressivement l'échelle s'éloigne de celle de la carte originale, la représentation des objets devient différente d'un objet à l'autre. Par ailleurs le niveau d'observation est fonction de l'échelle et il a une influence sur le concept et la définition des phénomènes, sur la nature de leur implantation et sur leur nombre.

Routes et chemins

Quelle que soit l'échelle, l'implantation demeure linéaire exception faite de certains aménagement routier complexes, échangeurs d'autoroutes, par exemple, dont la représentation devient ponctuelle à petite échelle.

Au grandes échelles, le critère de base est la viabilité qui s'exprime

- soit en termes d'ordre: grande circulation, viabilité excellente, bonne, moyenne, incertaine;
- soit en termes quantitatifs: nombre de chaussées, de voies ou largeur de la route.

La largeur d'une route peut d'ailleurs être exprimée de deux façons:

- la largeur de la partie utile qui conditionne le nombre de voies, les possibilités de circulation, la vitesse et la fluidité du trafic;
- la largeur de l'emprise totale, comprenant les fossés, les accotements, et éventuellement, le terre-plein central.

En pratique, les routes praticables aux véhicules à moteur sont représentées par deux traits parallèles noirs et les différentes classes sont obtenues, non seulement par

une variation d'écartement, mais également par des différences d'épaisseur des traits; pour les autoroutes, un trait médian met en évidence l'existence de deux voies séparés. Une route impraticable ou en construction est indiquée par des traits discontinus.

A moyenne échelle, les deux critères précédents sont toujours prédominants, mais il devient intéressant de faire apparaître, de façon claire et évidente, l'importance des relations assurées; comme la place est limitée, l'encombrement des signes est réduit par suppression des traits forts, certaines catégories sont fusionnées mais la simplification nécessaire du signe est compensée par l'apparition de surcharge de couleur symbolisant le nouveau critère. Ici la couleur introduit une relation d'ordre, conventionnellement admise, un rouge vif mettant en évidence le réseau principal, un jaune ou un rouge moins intense (tiretés, grisé ou pointillé) le réseau secondaire.

A petite échelle, le seul critère retenu est celui de l'importance des relations; aussi se limite-t-on le plus souvent à trois catégories, en opérant une sélection et une élimination importante, dans le réseau de relations locales. L'allégement et l'amenuisement obligatoire des signes fait renoncer à la représentation à deux traits fins pour passer au trait unique d'épaisseur variable, généralement rouge (voir le tableau 2.3).

Chemins de fer

La classification des chemins de fer est relativement plus simple car le nombre de catégories est réduit et les critères sont strictement quantitatifs: d'une part le gabarit des voies, voies normale, voies étroites, d'autre part nombre de voies. A toutes les échelles, l'implantation est linéaire et le signe imprimé le plus souvent en noir.

A grande échelle, où la place n'est pas limitée, le nombre de traits peut indiquer le nombre de voies tandis que l'épaisseur du trait varie selon le gabarit; cependant cette solution conduirait à un encombrement excessif des voies multiples et dès l'échelle de 1/50000, on doit abandonner ce principe par des traits uniques et le nombre de traits perpendiculaires accolés indiquant sans ambiguïté le nombre de voies.

Canaux et Rivières

L'implantation des canaux est toujours linéaire et leurs signes conventionnels imprimés en bleu. Mais seulement la largeur d'une rivière n'est pas toujours constante, d'où justifie la nécessité de différencier les canaux et les rivières[2].

On peut définir deux critères de classement: qualitatif selon que le canal est navigable ou non, quantitatif en fonction de sa largeur *utile*, de berge à berge.

A l'échelle de 1: 25 000, tout canal de largeur égale ou supérieur à 10m peut être représenté par deux traits bleus séparés par un espace teinté en bleu clair; les dimensions vraies sont respectées et la représentation est homométrique. Les canaux de moins de $10m$ sont uniformément représentés par un trait bleu unique.

Aux échelles inférieures, on a généralement une abstraction du critère dimensionnel, malgré des variations qui sont sensibles à moyenne échelle (de 4 à 25 m) et on se limite à deux signes. Le signe le plus large est attribué au canal ou rivière navigable. On considère, en effet, que le canal navigable fait partie du réseau des voies de communication.

[2] voir la section 4.11

A partir du tableau 2.3 de la page 28, on peut établir un tableau 2.4 de comparaison à la page 32.

Le tableau 2.4 de la page 32, établi à partir des spécifications actuelles des cartes IGN à l'aide de Monsieur Weger[3], fournit pour les routes, et chemins de fer les données numériques suivantes:

- E: l'échelle correspondante.
- D_r: dimension moyenne réelle pour chaque catégorie (exprimé en mtre).
- D_c: largeur totale du signe conventionnel correspondant.
- $D_r \times E$: encombrement réel que le largeur D occupe sur la carte à l'échelle E.
- r : coefficient d'amplification moyen résultant, et on a:

$$r = \frac{D_c}{D_r \times E} \tag{2.1}$$

Problème d'encombrement de signe

Pour donner une idée plus précise des variations de dimensions des signes conventionnels en fonction de l'échelle et de leurs conséquences sur l'homogénéité de la représentation, on peut voir le tableau 2.4 de la page 32.

Ce tableau montre que l'échelle de la représentation est variable d'un objet à un autre, et ce dans des proportions qui différent selon l'échelle. On constate également que l'amplification, pour une échelle donnée, s'effectue selon deux gradations différentes pour les signes à trait unique et pour les signes à traits multiples. Il existe toujours une disproportion entre les détails situés de part et d'autre part de la césure.

2.6 Placement des toponymes

Evoquons rapidement le placement de toponyme dans le cadre routier.

Les cartes sans toponyme (muettes) ont l'avantage de la clarté et de la lisibilité car les écritures sont plus encombrantes que les symboles. Mais le toponyme est le plus sûr moyen d'identifier un objet avec certitude; les écriture seront donc un élément indispensable de l'information géographique.

Comme toujours dans le problème de cartographique, la mise en place des écritures obéit à des règles qui sont les fruits d'une longue expérience mais qui doivent toujours être respectées.

La manière de disposer les écritures contribue dans une large mesure à la qualité d'une carte non seulement en précisant sans ambiguïté la localisation et la nature de la route, mais encore en participant à l'effet esthétique d'ensemble. Une harmonie générale est à rechercher, c'est à dire avec un minimum de goût et de sens artistique.

Les règles sont différentes selon qu'il s'agit de noms dits à position ponctuelle ou à disposition linéaire ou zonale.

[3]Ecole Nationale de Science Géographique, département cartographie

Objets			Échelle					
Type	D_r		1:25 k	1:50 k	1:100 k	1:250 k	1:500 k	1:1M
Voies ferrés +2 voies	7.5 m	D_c	0.7	0.3	0.4	0.3	0.3	0.3
		$D_r \times E$	0.3	0.15	0.075	0.03	0.015	0.0075
		r	**2.3**	**2**	**5.3**	**10**	**20**	**40**
Voies ferrés 1 voies	4.5 m	D_c	0.3	0.3	0.4	0.3	0.3	0.3
		$D_r \times E$	0.18	0.09	0.045	0.018	0.009	0.0045
		r	**1.67**	**3.3**	**8.9**	**16.7**	**33.3**	**89.9**
Auto-route	28 m	D_c	1.18	1.18	1.40	1.12	0.99	0.70
		$D_r \times E$	1.12	0.56	0.28	0.11	0.055	0.0275
		r	**1.05**	**2.11**	**5**	**10**	**18**	**25.5**
Route à 2x2v séparés	28 m	D_c	1.18	1.18	1.04	1.04	0.99	0.70
		$D_r \times E$	1.12	0.56	0.28	0.11	0.055	0.0275
		r	**1.05**	**2.11**	**3.7**	**9.48**	**18**	**25.5**
Route à +3 voies	17.5 m	D_c	0.98	0.98	0.99	0.76	0.55	0.40
		$D_r \times E$	0.7	0.35	0.175	0.07	0.035	0.0175
		r	**1.4**	**2.8**	**5.7**	**10.8**	**15.7**	**22.9**
Route à 2 voies larges	13 m	D_c	0.74	0.74	0.99	0.76	0.55	0.40
		$D_r \times E$	0.52	0.26	0.13	0.052	0.026	0.013
		r	**1.4**	**2.8**	**7.6**	**14.6**	**21.1**	**30.8**
Route à 2 voies étroites	7 m	D_c	0.55	0.55	0.64	0.46	0.3	0.3
		$D_r \times E$	0.28	0.14	0.07	0.028	0.014	0.007
		r	**2.0**	**3.9**	**9.1**	**16.4**	**21.4**	**42.8**
Route -5m	4.5 m	D_c	0.55	0.55	0.64	0.46	0.25	0.2
		$D_r \times E$	0.18	0.09	0.045	0.018	0.009	0.0045
		r	**3**	**6.1**	**14.2**	**25.6**	**27.8**	**44.4**
Autre route étroite	4 m	D_c	0.55	0.55	0.55	0.2	0.2	0
		$D_r \times E$	0.16	0.08	0.04	0.016	0.008	0
		r	**3.4**	**6.8**	**13.6**	**12.5**	**25**	
Route irrégulière	4 m	D_c	0.55	0.55	0.55	0.2	0.2	0
		$D_r \times E$	0.16	0.08	0.04	0.016	0.008	0
		r	**3.4**	**6.8**	**13.6**	**12.5**	**25**	
Chemin d'exploitation	2.5 m	D_c	0.2	0.2	0.2	0.2	0.2	0
		$D \times E$	0.1	0.05	0.025	0.01	0.005	0.0025
		r	**2**	**4**	**8**	**20**	**40**	
Sentiers	1 m	D_c	0.2	0.2	0.2	0.2	0.2	0
		$D_r \times E$	0.04	0.02	0.01	0.004	0.002	0.001
		r	**5**	**10**	**20**	**50**	**100**	
Rivière navigable			vrai grandeur					
Rivière navigable -10m	6 m	D_c						
		$D_r \times E$						
		r						

Tableau 2.4: Tableau de comparaisons des signes conventionnels: D_c et $D_r \times E$ sont en mm

2.6.1 Ponctuel

Par exemple, le nom d'un carrefour ou d'une ville est à disposition ponctuelle. Le placement est fonction de la place disponible.

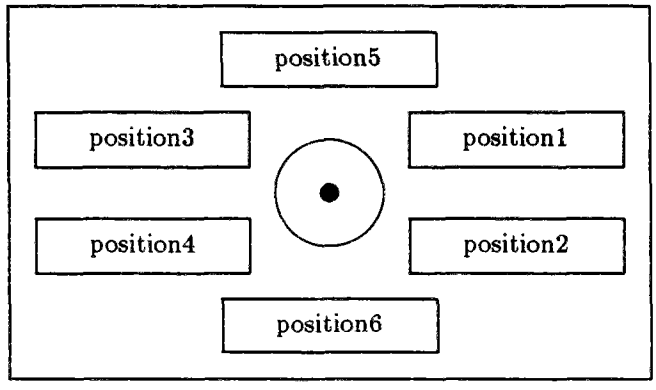


Figure 2.1: Positions préférentielles

1. Le nom est placé le plus près possible de l'objet à désigner et de préférence, à droite dans le sens de lecture.

2. On évite la superposition des noms et des détails planimétriques au détriment de l'emplacement idéal suivant l'ordre préférentiel indiqué par la figure 2.1:

   position1 > position2 > position3 > position4 > position5 > position6

3. Quel que soit sa longueur, un toponyme en un seul mot ne doit pas être tronçonné pour être disposé sur plusieurs lignes mais on peut placer sur deux lignes les noms de lieux habités composés de plusieurs mots en faisant la coupure après le premier mot (article non compris) et avant le séparateur s'il existe.

4. Les noms des objets situés d'un seul coté par rapport à une rivière, une route, une voie ferrée, une frontière ou toute ligne planimétrique importante, seront inscrits du même coté.

2.6.2 Linéaire

Le fait que nous sommes dans un cas linéaire et que la largeur du signe conventionnel des routes soit constante nous simplifie la tâche.

- Les noms de la route sont placés le long de la route en suivant l'orientation et le tracé de la route en faisant abstraction des petites sinuosités et en évitant qu'ils soient séparés de l'objet désigné.

- *Direction*: les noms sont placés de préférence au dessus de la ligne, à une distance à peu près égale à la moitié du corps ou un peu plus si le nom comporte des lettres descendantes;

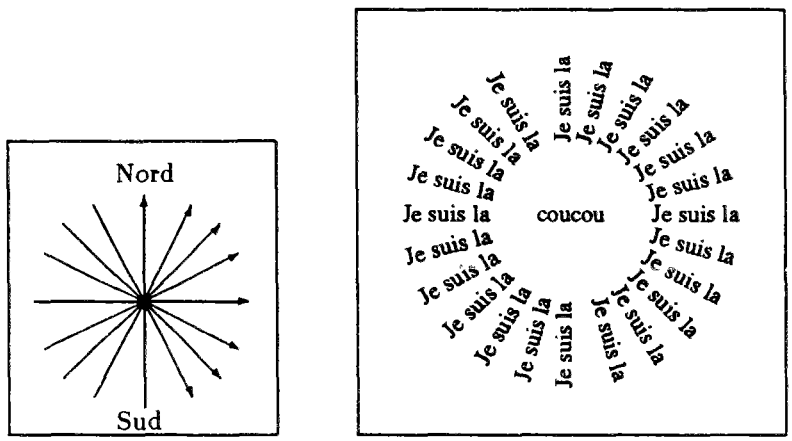


Figure 2.2: Les directions de placement de toponymes

- La disposition générale doit être telle que en regardant la carte de face, le Nord étant en haut, on puisse lire le nom, en tournant au besoin un peu la tête à gauche ou à droite, ce qui conduit aux sens d'écriture indiqués par la figure 2.2
- On répète le nom autant de fois qu'il est nécessaire pour suivre le parcours de la route par un espacement définit, et sans ambiguïté en particulier après les carrefours, sauf s'il n'y a pas assez de place, et compte tenu éventuellement du pliage de la carte.

En définitive, la stricte application des principes précédents n'est pas suffisante pour réaliser une bonne disposition car il faut, en plus, tenir compte de l'effet combiné de toutes les écritures et éviter les effets disgracieux résultant de groupements trop denses ou d'espaces vides ou d'apparition de figures géométriques attirant l'attention.

2.7 Un critère de résolution de problème

Les éléments ou les informations ne sont pas au même niveau, on doit donc les traiter selon leur importance. Pour généraliser un réseau routier, il vaut mieux commencer par généraliser le point le plus délicat: les carrefours.

On définit le

carrefour élémentaire : un seul noeud auquel se rejoignent des arcs; voir la figure 6.21 de la page 122.

carrefour complexe : au moins deux noeuds auxquels se rejoignent des arcs; c'est une suite de carrefours élémentaires qui sont proches l'un et l'autre selon le critère de lisibilité. Par exemple, sur la figure 1.1: **S1** est proche de **S3**, **S4** est proche de **S1**.

La méthodologie est de simplifier et décomposer le problème. Soit un carrefour complexe:

Carf-c = (carf-*e1*, carf-*e2*, ..., carf-*en*)

dont les carf-*ei* sont des carrefours élémentaires (avec un seul noeud) composant le Carf-c en raison des distances proches qui les séparent.

- On essaye de décomposer le problème en sous-problèmes moins complexes en traitant les carf-*ei* séparément:

  Généraliser[Carf-c(carf-*e1*, carf-*e2*, ..., carf-*en*]
  $\overset{\text{décomposer}}{\Longrightarrow}$ Généraliser[carf-*e1*]
  $+^{\Re}$ Généraliser[carf-*e2*]
  $+^{\Re}$ ...
  $+^{\Re}$ Généraliser[carf-*en*]

  où : $+^{\Re}$ n'est pas à prendre au sens d'une réunion simple, mais d'une réunion Relative. Il faut combiner les généralisations de chaque carrefour:

  Généraliser[carf-*e1*], ..., Généraliser[carf-*ei*]

  tout en respectant leur relations et leur structures internes et éventuellement externes avec leurs voisins non carrefours.

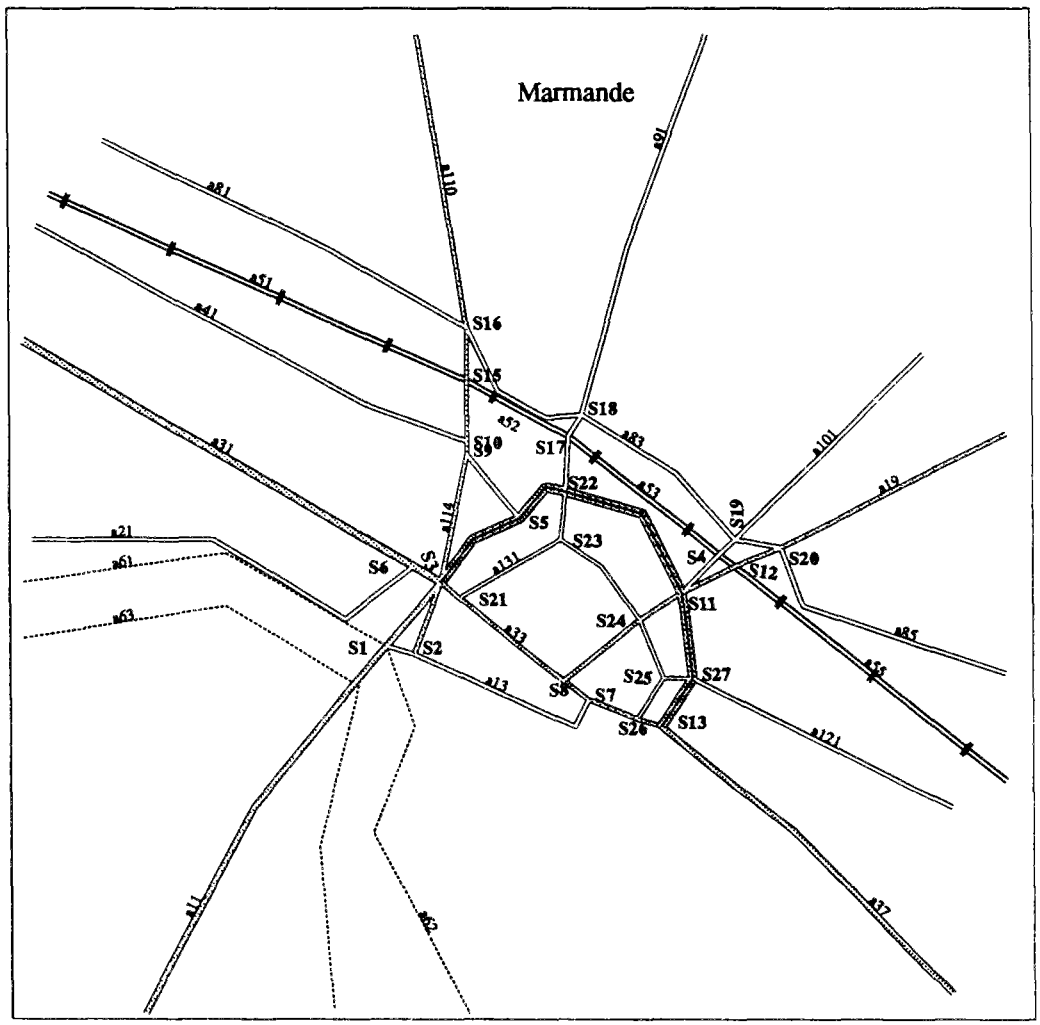
- Les sous-problèmes ne sont pas forcément toujours plus simples, donc on doit traiter en priorité les sous problèmes qui pourraient être plus complexes que le problème du départ. Par exemple, l'importance et la complexité d'un carrefour sont beaucoup plus grandes que celles d'un arc ou un axe tout seul. Pour généraliser un axe-routier Axe1(arc*1* arc*2* ... arc*n*) dont les arcs appartiennent à des carrefours carf*1*, carf*2*, ..., carf*i* respectivement.

  On décompose d'abord le problème en sous-problèmes pour traiter les arcs respectivement.

  Généraliser[Axe1(arc*1* arc*2* ... arc*n*)]
  $\overset{\text{décomposer1}}{\Longrightarrow}$ Généraliser[arc*1*]
  $+^{\Re}$ Généraliser[arc*2*]
  $+^{\Re}$ ...
  $+^{\Re}$ Généraliser[arc*n*]
  $\overset{\text{décomposer2}}{\Longrightarrow}$ Généraliser[carf*1*(arc*1*)]
  $+^{\Re}$ Généraliser[carf*2*(arc*2*)]
  $+^{\Re}$ ...
  $+^{\Re}$ Généraliser[carf*i*(arc*n*)]

Par la participation d'un arc à un carrefour, le traitement de cet arc revient à traiter le carrefour correspondant. Donc on se trouve à une décomposition d'un problème en *sur-problèmes.*

On pourrait dire que le processus de la G.C. est un ensemble de chaînes de décomposition en sous-problèmes et en même temps en sur-problèmes. Pour qu'on puisse exploiter les connaissances, le système doit savoir faire des requêtes à la demande. Pour la résolution de ces buts, on utilise des *règles*, leur rôle est de produire des nouvelles connaissances à partir de celles déjà fournies.



Echelle initiale: 1:25 000. Ci-dessous les reductions a la facon photocopieuse

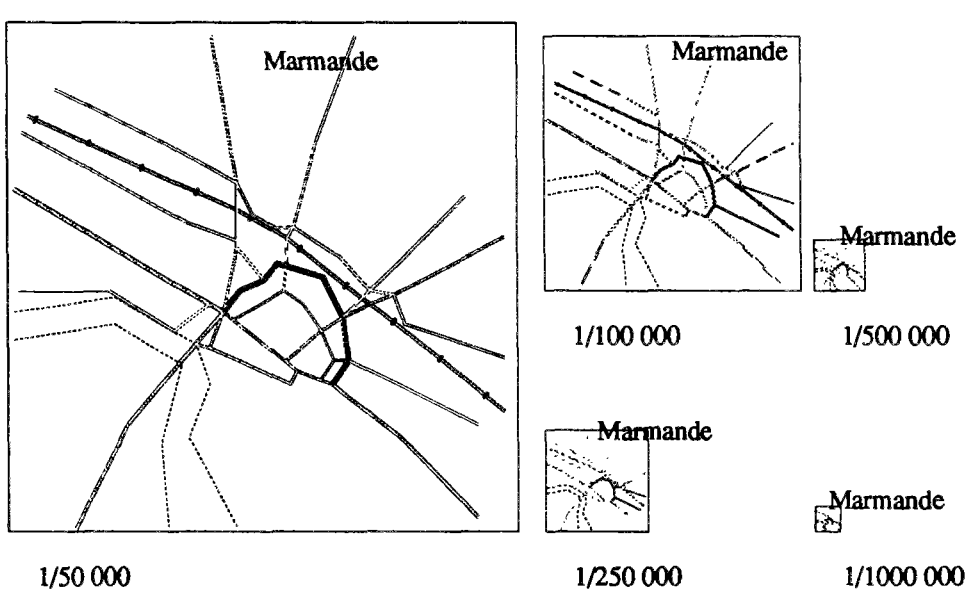
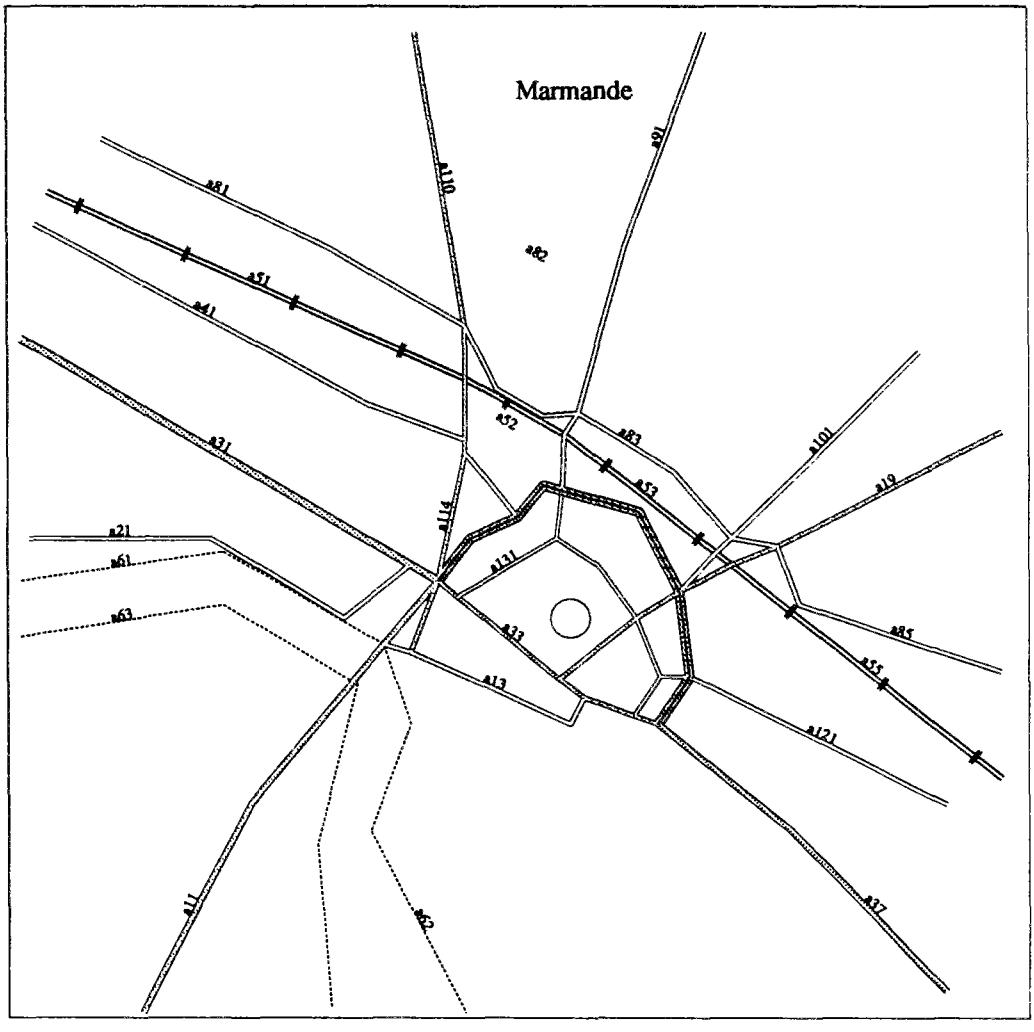


Figure 2.3: Réduction de carte à la façon photocopieuse



Echelle initiale: 1/25 000. Ci-dessous, les reductions sans generalisation

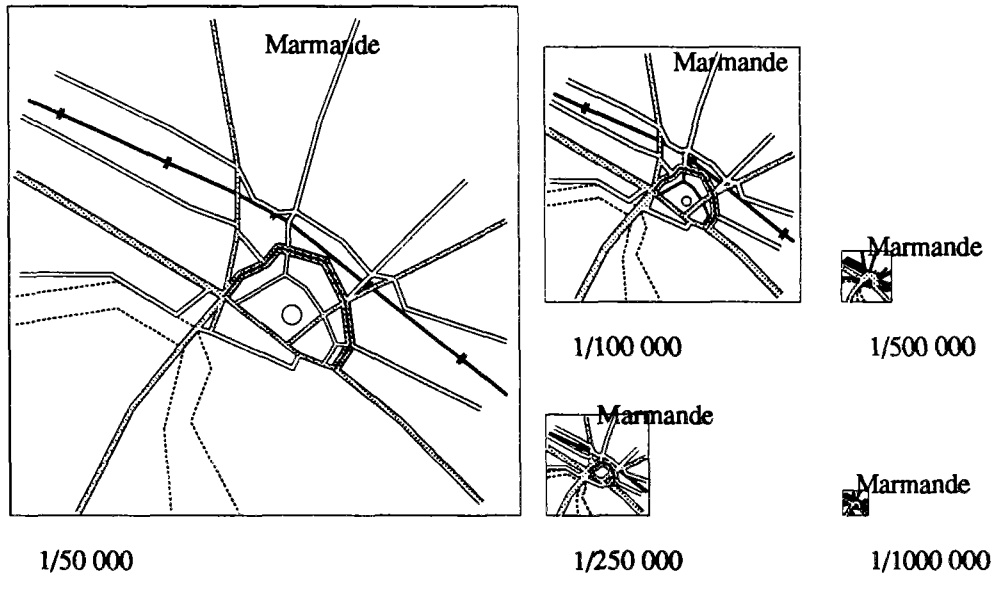


Figure 2.4: La carte de la ville Marmande: les réductions en prenant les signes conventionnels correspondant sans généralisation

Partie II

Outils Informatiques

Chapitre 3

Choix de modes de représentation des connaissances

3.1 Caractéristiques des systèmes experts

Le problème de la généralisation est tellement complexe, dit Mackaness[21], *qu'une méthode basée sur des connaissances heuristiques serait peut être le seul moyen qui puisse résoudre quelques problèmes centraux dans un tel processus.*

On espère contourner cette difficulté partiellement en faisant appel à la méthodologie des systèmes-experts. Un cartographe humain est capable d'identifier les problèmes posés en une première étape, il dispose ensuite des méthodes pour les résoudre. Pour qu'un système imite un cartographe humain en identifiant et proposant une résolution au problème posé, il doit aussi, comme un cartographe, avoir des yeux pour identifier, un cerveau pour réfléchir, analyser et faire des décisions, et des mains pour réaliser ses buts. Il doit comme les humains, posséder:

- des connaissances en cartographie,
- une méthode pour structurer ces connaissances,
- la capacité d'exploiter les connaissances pour pouvoir prendre des décisions.

3.1.1 Composants de base d'un système expert

Un système informatique *intelligent* comprend en général deux parties: une *base de connaissances*, et un *programme* qui opère sur la base. Plus précisément on peut dire qu'il est composé de trois éléments principaux: une base de faits, une base de connaissances, un moteur d'inférence.

Les systèmes experts se différencient principalement:

- d'un point de vue externe, par le langage de représentation des connaissances qu'ils utilisent. La qualité de ce langage définit la puissance d'expression et la formation de situations complexes.

- d'un point de vue interne, par la démarche déductive qu'ils propose. Cette démarche englobe les particularités, les stratégies de contrôle et les modes de raisonnement du moteur d'inférence. Il est clair que le langage et le moteur sont interdépendants.

Suivant le mode de représentation des connaissances choisi, et la logique du moteur d'inférences, un fait élémentaire peut être:

- un nom symbolique
  Ex: **intersecter** (formule atomique du calcul des propositions)
  Ex. (une Route1 **intersecte** la Route2)

- un quadruplet < objet attribut comparateur valeur >
  (formule atomique du calcul des propositions avec variables globales)
  ou (formule atomique du calcul des prédicats sans variable)
  Ex: (Route1 est de largeur $\geq$ 7m);

- un couple < prédicat variables >
  (formule atomique du calcul des prédicats)
  Ex: (Intersecter ?route1 ?route2)

Le raisonnement des systèmes experts est soit basé sur la logique des propositions, soit sur la logique des prédicats: on parle alors respectivement de système sans variables (d'ordre 0), et de système avec variables (d'ordre 1).

Dans l'idéal, un tel système doit être facile à comprendre et à utiliser, et suffisamment souple pour pouvoir être adapté à des domaines divers.

En résumé, les différents éléments d'un système expert sont les suivants:

La base de faits :

Elle contient à tout moment des connaissances assertionnelles: la collection des informations, touchant le cas particulier traité, données par l'utilisateur ou obtenues par le système. C'est la mémoire de travail du système expert; temporaire, elle est modifiée au fur et à mesure de la progression du raisonnement, et éventuellement vidée lorsqu'une consultation se termine.

La base de connaissances :

Elle rassemble la connaissance et le savoir-faire du spécialiste dans son domaine: description des objets et de leurs relations, cas particuliers, points de vue divers sur un même problème, stratégies de résolution et conditions d'application. Deux sortes de connaissances sont distinguées dans la section 3.2.

Le formalisme le plus communément répandu pour représenter la base de connaissances est celui des règles de production. On l'appelle alors base de règles. Les détails sont dans la section 3.3. Notre système est basé principalement sur les règles de production.

Le moteur d'inférence :

C'est un programme qui met en oeuvre des mécanismes généraux d'interprétation des connaissances d'un domaine particulier. Généralement, l'action du moteur provoque des modifications de la base de faits et dans certains cas de la base de règles. Un moteur d'inférences est surtout caractérisé par la façon dont il utilise les connaissances.

Le mode d'inférence :

On distingue principalement deux sorte de stratégies de raisonnement: chaînage avant et chaînage arrière.

chaînage avant: raisonnement déductif : Dans ce mode, le moteur essaye d'obtenir tous les faits déductibles à partir des règles dont les conditions d'application sont vérifiées par les éléments de la base de faits. Chaque fois qu'une règle est déclenchée, le moteur utilise les nouveaux faits obtenus pour activer d'autres règles. Il s'arrête lorsqu'un but est atteint ou lorsqu'aucun fait ne peut plus être déduit. On utilise le chaînage avant lorsqu'on n'a pas une idée précise sur l'objectif à atteindre.

Ainsi, cette démarche convient à notre problème de généralisation cartographique où à partir d'un ensemble de "faits" (les données initiales concernant la carte d'échelle initiale), on déduit tous les "faits" finaux (les données finales d'échelle finale de la carte souhaitée). Dans cette optique, le système "sature" sa base de connaissances; il s'arrête lorsque plus rien ne peut être déduit.

chaînage arrière :

Dans ce mode, le moteur cherche à atteindre un ou plusieurs buts considérés comme des hypothèses ou des problèmes à résoudre. Pour cela, il essaye d'appliquer les règles qui "concluent sur ces buts". Lorsque l'on a un but précis à atteindre, le chaînage arrière s'avère judicieux.

Un cartographe qui veut savoir si un carrefour est éliminable, va mettre le système expert en chaînage arrière.

chaînage mixte :

Dans ce mode, il y a mélange des deux types de raisonnements décrits ci-dessus. Le chaînage avant peut appeler le chaînage arrière lorsque certains faits n'ont pas pu être déduits, ou au contraire, appel du chaînage avant par le chaînage arrière lors d'une résolution. Ce deuxième cas est très fréquent, et permet, lors de la vérification d'un but, de déduire des informations supplémentaire.

Fonctionnement du moteur d'inférence :

La stratégie de raisonnement d'un MI comporte trois phases consécutives:

- détection des règles intéressantes.
- sélection de la règle à appliquer;
- déclenchement de la règle retenue.

Selon le mode d'interprétation (avant ou arrière), ces phases n'auront pas les mêmes actions.

La puissance d'un système expert dépend principalement de la qualité de ses déductions et de la valeur des informations qu'il possède sur le domaine traité (son "expertise"). La puissance déductive est localisée dans le "moteur d'inférence" dont les techniques sont généralement bien maîtrisées. Par contre, on connaît fort peu de choses sur la constitution de bonnes bases de connaissances. C'est un art délicat connu sous le nom d'ingénierie de la connaissance; elle nécessite d'être structurée et organisée pour être exploitée. Cette acquisition des connaissances reste une tâche

longue et difficile. La base de connaissances dépend du domaine d'application. Généralement, au cours d'une consultation, les connaissances du domaine traité ne varient pas. Ainsi, la base de connaissances est fixe, elle reste inchangée au cours d'une exécution, et intéresse au premier chef le spécialiste du domaine puisqu'il a pour mission de la construire. Par contre, la base de faits est liée au cas particulier envisagé, elle correspond à la description du problème traité et évolue au cours du traitement et de la consultation.

3.1.2 Notre système, en résumé

Notre système essentiel comprend: voir la figure 1.2

- Un système expert:
  - un système de représentation de connaissances à base d'objets.
  - un moteur d'inférence avec variables, en chaînage avant, non monotone.
- une interface entre le système et l'utilisateur par l'interprète `Lisp`.
- un module de traduction des données en code PostScript.

Dans notre exemple de la généralisation cartographique, pour deux cartes à généraliser nous aurons la même base de règles, mais deux bases de faits différentes qui correspondent aux deux états géographiques et cartographiques. Souvent un cartographe procède, pour établir un processus de généralisation, par alternance entre l'évocation et la confirmation. En fonction des informations sous ses yeux, il évoque des actions possibles, puis par recherche du meilleur compromis entre la clarté, la fiabilité et l'esthétique, il cherche à infirmer ou à confirmer ses actions, ou à changer d'avis en prenant de meilleures décisions. Il peut donc déduire de nouvelles actions, ou au contraire éliminer les actions qu'il avait envisagées. En bref, un expert humain acquiert ou augmente sa connaissance en travaillant.

Si les ensembles "base de faits" et "base de connaissances" sont distincts, les unités de connaissances qui constituent la base de faits et ceux qui apparaissent dans les éléments de la base de connaissances sont conceptuellement les mêmes. Dans notre exemple sur la généralisation cartographique, ces unités, appelées "faits", pourraient être:

- l'échelle;
- le but de la carte;
- les éléments géographiques: les maisons, les routes, etc.
- les interprétations graphiques;
- les signes conventionnelles;

Une des tâches délicates lors de la conception d'un système expert est la détermination d'un formalisme adéquat pour représenter ces unités de connaissances. Les

domaines d'application privilégiés du système, ainsi que ses performances et ses possibilités déductives, sont directement reliés au choix de représentation des connaissances. Pour cette raison, la section suivante est consacrée aux principaux formalismes utilisés.

Avec les systèmes experts, on a réalisé de bons simulateurs du raisonnement humain. Ils ont montré toute l'importance de la connaissance humaine indispensable à un programme pour obtenir un embryon de raisonnement. Avec eux, d'importants travaux de recherche ont pris naissance, et se sont naturellement orientés vers les problèmes de représentation des connaissances en mémoire. Le grand nombre de formalismes différents existants aujourd'hui montre toute l'importance qu'attachent les chercheurs à définir la structure "idéale" pour modéliser la connaissance des experts. Des progrès réalisés dans ce domaine dépend de l'avenir des systèmes experts.

3.2 Difficulté de concevoir la connaissance

Pour résoudre un problème, un être humain raisonne sur des concepts abstraits qui modélisent les entités de l'univers du problème et qu'il interprète ensuite dans cet univers.

L'expert humain acquiert et augmente sa connaissance au cours de son travail en percevant de l'information: selon quels mécanismes cette information se structure t-elle et s'organise-t-elle, sans effort de sa part, en connaissances?

Faire reproduire le comportement du raisonnement humain par un programme suppose d'abord une étape d'abstraction, qui consiste à établir les spécifications des concepts devant être représentés. Ces spécifications sont ensuite traduites en expressions symboliques d'un langage de représentation pour former une base de connaissances qui peut être comme un modèle de l'univers du problème. Le concepteur d'une base de connaissances doit donc d'abord déterminer quelle est la connaissance à représenter, puis se demander comment l'organiser, comment l'exprimer et quelles contraintes lui imposer pour qu'elle soit en accord avec l'univers modélisé.

Si un concept représente une entité, une action ou un état faisant partie d'un certain univers, on peut le décrire de deux manières différentes [16], [15]:

- en *extension*, par énumération de tous les individus, ou référents, qui sont représentés par le concept,

  Si **Vc** représente le concept de *voies de communication*, **Vf** le concept de *voies ferrées*, et **Vr** celui de *voies routières*, l'extension de **Vc** contient celle de **Vf** et celle de **Vr**: l'énumération de toutes les voies de communication contient l'union de l'énumération de toutes les voies ferrées et de celle des voies routières.

- en *intension*, par énumération des propriétés communes à tous les individus représentés par le concept.

  A l'inverse de la manière précédente: l'intension de **Vc** est contenue dans celle de **Vf** et celle de **Vr**: les voies ferrées et les voies routières possèdent l'union des caractéristiques des voies de communication et de celles qui sont propres aux voies ferrées et aux voies routières respectivement.

C'est par l'*intension* qu'est décrit un objet dans les langages de classe (section 4.3).

Généralement, un concept n'est pas isolé mais en relation avec d'autres concepts. Il existe deux modèles de concept:

Variable	exemples					
forme	rectangle	carré	cercle	triangle	étoile	losange
dimension	$5mm^2$	$4mm^2$	$3mm^2$	$2mm^2$	$1mm^2$	$0.5mm^2$
valeur	noir: 1		gris: (0,1)		blanc: 0	
orientation	vers le haut	à gauche	vers le bas	à droite		
couleur	noir	bleu	vert	rouge	jaune	marron

Tableau 3.1: Les variables de la cartographie

- *modèle relationnel*, qui privilégie les relations: un concept n'existe que par l'ensemble des relations dans lesquelles il intervient; la base de donnée HBDS en est un exemple[3].

- *modèle structurel*: qui met l'accent sur l'*objet*, représentation d'un concept et de ses propriétés: la connaissance disponible sur un concept est regroupée au sein d'une même entité.

Les objets de la vie courante n'obéissent pas à des lois aussi rigoureuses que les objets mathématiques par exemple. Intégrer leurs différentes natures dans un formalisme de représentation pose de nombreux problèmes difficiles à résoudre.

Le transfert de la connaissance du spécialiste vers le programme, pour une application particulière, est une tâche longue et difficile qui constitue un véritable obstacle au développement des systèmes experts. Elle nécessite une étroite collaboration entre deux corps de métiers parfois très éloignés: le domaine d'expertise et l'informatique. Pour que l'opération soit concluante, les personnes engagées dans le projet doivent faire preuve d'une grande ouverture d'esprit.

Il est difficile, voire impossible, de représenter toute la connaissance d'un domaine dans un formalisme unique. Prenons deux propositions très communes des cartographes:

- *une route est une voie de communication.*

- *il faut que les représentations graphiques des objets soient claires et efficaces.*

On perçoit intuitivement qu'elles ne sont pas de même nature. L'une décrit un objet et le situe dans un groupe, qui n'est autre qu'un objet plus abstrait. L'autre décrit un caractère général du travail qui permet de définir des critères et des hypothèses pour garantir une résolution efficace du problème. Nous dirons que la première est une connaissance conceptuelle, la seconde une méthode ou une connaissance experte ou encore heuristique. En effet, il existe deux groupes de connaissances:

conceptuelles : elles sont souvent bien définies et peuvent généralement être bien structurées et bien représentées par un réseau sémantique, un réseau de schémas, etc. Par exemple:

- Pour représenter graphiquement un phénomène concret ou abstrait localisé en un point de la surface terrestre, le cartographe dispose [1]:
  - en premier lieu, des deux dimensions du plan,
  - en second lieu et pour différencier les images, des variations de *forme, dimension, valeur, orientation, couleur.*

  Ces cinq composantes constituent les *variables visuelles* cartographiques (voir le tableau 3.1), elles permettent d'établir:
  - une *relation d'équivalence* permettant de discerner par la perception visuelle la famille des éléments appartenant à une même classe d'équivalence et éventuellement, de les isoler des autres éléments pour les grouper en une image globale unique faisant abstraction de tous les autres signes,
  - une *relation d'ordre* qui permet de différencier les familles de catégories différentes et de les classer dans un ordre qui s'impose naturellement et sans ambiguïté. Cet ordre peut être intuitif: *ceci est plus grand, ou plus important, que cela*, mais il peut aussi exprimer spontanément un rapport numérique approximatif: *ceci est environ le triple de cela.*
- une autoroute est une voie rapide de communication de transport, à double sens, avec un séparateur central; elles joue un rôle important pour les cartes topographiques, donc elle doit être représentée de couleur vive; par convention, elle est représentée par un trait au milieu et deux traits au bords séparés par deux espaces de couleur rouges sur des cartes à échelle moins de 1: 250 000, et par deux traits séparés par un espace de couleur rouge à partir de l'échelle 1:500 000.
- Deux routes non collées devraient être séparées par un espace de telle sorte que l'oeil humain puisse les distinguer.

Ce type de connaissance est partagée par les experts du domaine et figure dans les manuels, revues spécialisées, etc.

heuristiques : un groupe des connaissances touchant aux principes de cheminement du raisonnement, aux prises de décision, aux stratégies de résolution, aux démarches diagnostiques de l'expert, empiriques par nature, acquises par l'expert au cours de ses années de travail.

Ce type de connaissance est bien sûr la plus difficile à saisir, du fait que les experts eux-mêmes éprouvent de grandes difficultés à la cerner. Par exemple:

- un toponyme doit être placé le plus près de l'objet à désigner, mais à quel degré de proximité?
- pour les règles de sélection des objets facultatifs, la section 6.2.2 a défini une formule de sélection quantitative des éléments à maintenir sur la carte finale, mais en laissant toute la liberté de conserver tel objet plutôt que tel autre, et face à un grand nombre des éléments, une grande diversité de

décisions seraient prises par des cartographes de personnalités, de formations, d'expériences différentes.

Intégrer leurs différentes natures dans un formalisme de représentation pose donc de nombreux problèmes difficiles à résoudre:

- Les *modalités*: toutes les informations n'ont pas le même statut. Elles sont intangibles ou modifiables, certaines ou incertaines, valides ou périmées. Par exemple, dans la généralisation cartographique:
  - *le nom de toponymie doit être placé le plus près possible de l'objet à désigner,*
  - *Certains cartographes pensent qu'il ne faut pas trop charger la carte.*

  Le nom d'un toponyme sera placé dans des positions différentes, et la densité d'une carte est donc variable selon les différents cartographes.
- Le *type de propriété* et le *partage de propriété*: une connaissance comme *une autoroute constitue un phénomène essentiel* n'a pas de valeur universelle. Elle exprime le fait qu'en général une autoroute est un phénomène essentiel à l'échelon national ou international, bien qu'elle n'ait pas beaucoup plus d'importance dans le cadre de la commune qu'elle traverse que le moindre chemin d'exploitation. Il faut donc attribuer aux divers éléments des valeurs qui dépendent de l'échelle spatiale représentée et dont la généralisation tient compte.

  Une famille d'objets possède globalement des propriétés spécifiques, qui peuvent être différentes, voire en contradiction, avec les propriétés particulières à un objet de la famille.

  Par conséquent, il est toujours difficile, sinon impossible, d'établir des conditions nécessaires et suffisantes d'appartenance d'un objet du monde réel à une classe.
- Les *connaissances imcomplètes*: la connaissance disponible sur un objet est la plupart du temps incomplète, parce qu'elle est implicite et donc généralement oubliée dans la représentation, ou encore difficile à transmettre.
- L'*évolutivité*: les connaissances changent au fur à mesure que l'univers dans lequel elles sont utilisées se modifie. Des informations sont créées, modifées ou détruites. Certaines d'entre elles étant en relation, la mise à jour des informations périmées doit être assurée. Par exemple, si l'*échelle* change, les tailles des symboles doivent être changées de telle façon qu'ils soient lisibles. D'autres informations changent de type, quand ce n'est pas le type lui-même qui se transforme: par exemple: on distingue une douzaine de classes de routes et chemins sur la carte de France à 1/25 000 et quatre seulement sur la carte à 1/500 000 où les routes principales (classement qualitatif) sont regroupées dans un signe unique; les routes de plus de 7 m et de moins de 7 m (classement quantitatif) sont différenciées aux échelles plus grandes.

  Dans ce cas, il faut prévoir le réajustement des objets préexistants afin que l'univers conserve sa cohérence.

Donc, la mission de la construction de la base de connaissances est la suivante:

- identifier les concepts clés du domaine et les différentes relations qui les lient.
- dégager les structures logiques du raisonnement, et apprécier l'opportunité d'utiliser ou non un raisonnement flou.
- déterminer les meilleures façons de représenter et de traiter les concepts dégagés (définir le langage d'expression).
- écrire dans le langage défini toute la connaissance utile de l'expert.
- exhiber une séries d'exemples typiques du domaine (ni trop simples, ni trop compliqués) permettant de valider les étapes précédentes.

3.3 Quelques modes de représentation des connaissances

La qualité de la représentation des connaissances est déterminante pour la performance d'un Système Expert. Pour réaliser une bonne représentation des connaissances dans un système expert, il faut prendre en compte les points suivants:

extensibilité : les modèles envisagés doivent être souples pour permettre une modification aisée du système;

simplicité : les concepts retenus ne doivent pas être trop complexes, de manière à ce que l'expert non informaticien puisse aisément transmettre son savoir au système;

caractère explicite : la connaissance doit être explicite, c'est le facteur important dans la recherche des erreurs et la justification du système.

Il existe de nombreux formalismes de représentation des connaissances. L'idée principale de la méthodologie est que les connaissances doivent être stockées de manière très accessible pour disposer de nombreux points d'entrée, mais aussi afin d'utiliser ces connaissances pour faire à la fois de la vérification et de la génération des solutions demandées.

Les connaissances doivent être structurées, prototypées, testées, évaluées, elles doivent aussi être adaptées pour les applications et les algorithmes existants qui sont assez souvent dépendants des types de données.

Il existe principalement les modes de représentations suivantes:

- les représentations procédurales classiques dans la programmation classique avec des structures de données classiques.
- les représentations à base de règles telles que la programmation logique et les règles de production. Les systèmes de règles de productions sont issus des travaux sur les grammaires formelles et les règles de réécriture; les exemples sont les langages à base de règles et les moteurs d'inférences, tels que OPS et ses variantes. Ils sont utilisés pour représenter des connaissances déductives.

- les représentations orientées objets: dans un sens très large, les réseaux sémantiques, les objets, les acteurs, les frames et les représentations hybrides sont dans cette catégorie.

  Il s'agit d'un ensemble d'informations structurées comportant à la fois des codes, et à la fois des méthodes d'accès et de manipulation des données; on dispose de nombreux mécanismes de propagation et d'héritages de propriétés.

  Les langages orientés objets actuels tentent une synthèse entre les structures des *frames*, les mécanismes d'héritages des réseaux sémantiques, et les techniques de contrôle par transmission de messages. Smalltalk en est le pionnier.

Deux points communs à toutes ces représentations de la connaissance en IA sont l'*accès associatif* et le *non-déterminisme*.

- *accès associatif*: on a accès à des connaissances non pas par leur adresse en mémoire, mais par des traits, des caractéristiques de leur contenu, c'est à dire par *filtrage* ou *unification*.

- *non-déterminisme*: il est lié au caractère déclaratif des connaissances, c'est-à-dire à la possibilité de les fournir sans être contraint de prévoir explicitement tous leurs agencements et toutes leurs utilisations possibles.

Il est difficile de représenter dans un formalisme objet des connaissances qui se traduisent plus naturellement en formules logiques ou encore en règles de production. Plutôt que de rester prisonnier d'un formalisme unique, il est commode d'en faire coexister plusieurs dans un même système. Dans les paragraphes suivants et dans le chapitre 4, on va essayer d'analyser les caractéristiques de ces différents modes existants, et montre les intérêts d'adopter finalement le mode hybride pour représenter les connaissances géographiques et cartographiques de façon procédurale, par règles, et par objets; c'est-à-dire: raisonner et déduire des connaissances avec des règles, représenter des règles et des objets réels par des objets compacts qui sont dotés des algorithmes, des procédures classiques, et des méthodes d'accès et de manipulation.

3.4 Procédural ou déclaratif?

Un système est *procédural* s'il sait quelle connaissance il utilise, et comment les utiliser, c'est à dire trouver les faits pertinents et ensuite réaliser des bonnes inférences.

Un programme du type procédural considère que les données à manipuler sont liées directement à leurs mécanisme d'interprétation; les systèmes procéduraux ayant comme principal souci de suivre une ligne naturelle dans le raisonnement.

Lorsque le processus intellectuel, par lequel un humain évalue une situation ou prend une décision, est précisément modélisé (il n'en reste plus de choix non défini), il est relativement aisé de le programmer par un processus procédural, par exemple calculer la perpendiculaire à une droite passant par un point donné, chercher tous les points dans un cercle d'un diamètre défini autour d'un point donné. C'est aussi le cas, par exemple, dans des domaines tels que la comptabilité, le calcul scientifique ou la commande de machines-outils.

Dans certains domaines importants d'application, tels que le diagnostic médical, l'orientation scolaire, la justice, la généralisation cartographique qui nous concerne ici,

il est largement fait appel à des connaissances éparses, parcellaires, souvent d'origine expérimentale ou heuristique. Dans la généralisation cartographique, comme dans bon nombre d'activités intellectuelles qualifiées, relevant tant des sciences de l'ingénieur que des sciences humaines, des sciences sociales ou des sciences de la vie, le savoir-faire des experts humains n'est pas suffisamment structuré pour que l'on dispose d'algorithmes représentatifs de ces activités.

Dans ces domaines, le savoir-faire des spécialistes semble alors plutôt représentable comme un ensemble d'unités de travail, chacune étant appropriée pour une classe de situations éventuelles. Plusieurs unités peuvent avoir trait à la même classe de situations. Chaque unité, appelée *règle*, décrit une étape possible du raisonnement des spécialistes dans le domaine considéré. Dans ces domaines il est important de pouvoir aisément compléter, réviser l'ensemble des unités de savoir-faire. Ceci amène à les considérer comme un type particulier de données qui seront exploitées par un programme relativement général comme un **moteur d'inférence** ou une *machine déductive.*

Une représentation où les connaissances sont considérées comme des données manipulées par un interprète général, comme dans les systèmes à règles de production, est dite *déclarative.* Dans ce cas, les connaissances n'ont pas un caractère opératoire et il n'est fait aucune hypothèse sur la façon de les utiliser.

A l'inverse des premiers systèmes déclaratifs qui pour obtenir un résultat faisaient une recherche "aveugle" sans stratégie efficace, les systèmes procéduraux, dont les procédures sont désignées chacune par un nom et s'appellent naturellement par l'intermédiaire de leurs nom, dirigeaient exactement le contrôle en éliminant toute tentative inutile. Seulement en contre-partie, la complexité pouvait devenir importante, rendant toute modification délicate. Des connaissances procédurales ont un caractère opératoire et leur pertinence ne peut être jugée qu'après exécution du programme qui les manipule.

A l'inverse, la pertinence des connaissances peut être discutée *a priori.* La rédaction déclarative des règles dans les systèmes experts permet, en principe, de rédiger chaque règle (unité de savoir-faire) en ignorant l'existence effective des autres. On présume seulement que chaque règle introduite dans la base de connaissances comporte, quel que soit son auteur, des expressions de conditions qui permettront d'évaluer si la règle est applicable et donc, éventuellement, de l'appeler. On présume aussi que les faits introduits comme effets d'une règle peuvent contribuer à appeler d'autres règles. Le grand avantage d'une rédaction déclarative des règles est de permettre d'ajouter et de supprimer aisément des règles sans être obligé d'étudier les conséquences des ajouts et des suppressions; elle confère aux systèmes experts une meilleur modularité.

Il est très délicat de déterminer quel formalisme, déclaratif ou procédural, est le mieux adapté à la représentation des connaissances spécifiques d'un problème tel que la généralisation cartographique. Ce genre de comparaison fut à l'origine d'une polémique célèbre appelée *controverse déclaratif-procédural.* Elle a eu le mérite de faire progresser les travaux sur la représentation des connaissances en mettant en évidence les insuffisances et la complémentarité de chacune des méthodes. La controverse a finalement abouti à l'élaboration des langages de frames et des langages hybrides.

Notre point de vue est de bénéficier conjointement des avantages de chacun; tant qu'un processus est bien précis, on adopte la méthode procédurale. Par exemple, les calculs formels, des recherches des informations sur un objet structuré, tel que "quels sont les voisins d'une route". Tandis que pour les décisions délicates nécessitant des

confirmations et susceptibles d'être changées à un instant ou l'autre, tel que "une route doit être déplacée ou éliminée en cas de discussion de son avenir", on préférera prendre la méthode déclarative. C'est à dire que les décisions à prendre sont obtenues de façon déclarative, alors que les actions des décisions, une fois prises, sont exécutées procéduralement, l'action de suppression d'une route est vite faite par la suppression de son nom de la liste de toutes les routes vivantes.

3.5 Raisonner logiquement

Etre logique avec soi-même, la logique est d'abord une science du discours cohérent, c'est à dire qu'elle considère moins le contenu des propositions que les conditions de leur enchaînement. La logique classique joue un rôle majeur en représentation des connaissances, même si, historiquement, elle a plutôt été conçue pour modéliser le raisonnement. Son formalisme est dépourvu d'ambiguïté et permet de décrire des situations du monde réel par des formules concises.

La logique est caractérisée : d'une part, par un langage formé des assertions que l'on peut exprimer (les formules ou propositions du système logique formel), et d'autre part, par les lois régissant les entités manipulées, à savoir les règles déductives.

- son langage: un ensemble de formules; les règles de formation des formules, spécifient la manière de construire des expressions correctes du langage. Les formules sont construites à partir des atomes à l'aide des connecteurs ($\wedge$: "et", $\vee$: "ou", $\neg$: "non", $\Rightarrow$: "implique", $\perp$: "faux") et des quantificateurs pour la logique des prédicats ($\forall$: quantificateur universel *quelque soit*, $\exists$: quantificateur existentiel *il existe*).

- son système d'inférence: un moyen de former des jugements et de déduire des jugements à partir d'autres jugements. Il existe plusieurs système d'inférence dont les systèmes de déduction naturelle NK pour la logique classique, NJ pour la logique intuitionniste, NM pour la logique minimale[18]. En déduction naturelle, un jugement est une formule.

- sa sémantique: (notion de modèle, d'interprétation) qui traite du sens attribué à ces expressions selon ce que l'on appelle des interprétations: la sémantique régit la manière de comprendre les expressions du langage en vue d'applications déterminées.

Deux sortes de logiques sont distinguées selon leur langage: la logique des propositions et la logique des prédicats. Les formules du calcul des propositions ne prennent que les valeur booléennes *vrai* ou *faux* dans une interprétation (modèle, monde) donnée sans variable. Par exemple:

1. *La terre est ronde.*
2. *Les chinois sont les plus nombreux sur la terre.*
3. *Une carte est plate.*

La première proposition prendra toujours la valeur *vrai*; la seconde prendra pour l'instant la valeur *vrai*, et peut-être dans 20 ans, quand les Indiens auront réussi à

rattraper les Chinois, elle prendra la valeur *faux*; la troisième est *fausse* dans une interprétation où il existe des cartes en relief.

Par rapport à la logique des propositions, la logique des prédicats du premier ordre introduit deux notions importantes qui sont les variables et les quantificateurs. Les formules sont construites à partir des atomes (qui expriment des relations entre termes) à l'aides de connecteurs et de quantificateurs: $\forall$ (universel) et $\exists$ (existentiel).

3.5.1 Règles d'inférence en logique

Une règle d'inférence en logique est la représentation d'un procédé pour, à partir d'un ou plusieurs faits (formules), dériver d'autres faits. Par exemple:

- la règle d'inférence appelée *modus ponens*, à partir de deux faits respectivement de la forme **A** et $\mathbf{A} \rightarrow \mathbf{B}$, déduit le fait **B**, représenté par:

$$\frac{A \quad A \rightarrow B}{B}$$

Elle permet à partir d'assertions connues (vraies ou fausses) d'en déduire de nouvelles qui en dépendent. Le modus ponens est à la base de presque tous les systèmes experts avec un moteur d'inférence fonctionnant en chaînage avant. Tandis que les systèmes avec un chaînage arrière lisent le modus ponens à l'envers: pour établir **B**, si $\mathbf{A} \Rightarrow \mathbf{B}$ est vrai, alors établir **A**.

On a aussi des règles pour la conjonction:

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

- pour le calcul des prédicats, il existe des règles permettant de raisonner sur les quantificateurs[18, pages 123]:

$$I_\forall : \frac{A(t)}{\forall x A(x)} \qquad E_\forall : \frac{\forall x A(x)}{A(t)}$$

où I signifie *Introduction*, E *Elimination*. La *spécialisation universelle* $E_\forall$ est une règle d'inférence principale pour rendre compte d'un raisonnement en logique des prédicats. Elle consiste à substituer chaque occurrence de la variable x dite *libre* de $A(x)$[18] dans une formule $\forall x A(x)$ par un terme, soit t, on déduit le fait $A(t)$. Ainsi est éliminé $\forall$ (quelque soit).

- une autre règle d'inférence peu utilisée est la règle d'inférence appelée *modus tollens*, à partir de deux faits respectivement de la forme $\neg$ **B** et $\mathbf{A} \rightarrow \mathbf{B}$, déduit le fait $\neg$ **A**.

$$\frac{\neg B \quad A \rightarrow B}{\neg A}$$

3.5.2 La logique des prédicats

Le calcul propositionnel n'est pas suffisant pour représenter des connaissances complexes. Dès lors, on a recours à un système logique plus puissant: la logique des prédicats, qui permet d'exprimer des connaissances complexes avec rigueur.

Elle permet d'exprimer des formules dont la valeur de vérité dépend de paramètres. Par conséquent, elle permet de particulariser ou de généraliser des objets.

Un *prédicat* est une entité à laquelle on peut appliquer un ou plusieurs arguments, qui retourne une valeur booléenne fonction de la valeur de ces arguments.

Soit le prédicat *Grand* dans *Grand*(x) qui appliqué à la *Chine* donnera la valeur *vrai*, et avec la *France* prendra la valeur *faux*, dans l'interprétation chinoise.

Par rapport à la logique des propositions, la logique des prédicats introduit les *variables* et les *quantificateurs*, ce qui la rend capable de représenter et d'exploiter davantage de connaissances.

Une *variable* est un objet auquel on peut substituer n'importe quel élément constant. Par exemple, pour définir le prédicat **intersecter** entre deux routes, on écrira: **Intersecter(?x, ?y)** indiquant que "?x" et "?y" s'intersectent (?x, et ?y étant pris ici en tant que variables). En pratique les formules bien formées du type:

$$\forall ?x\ (A(?x) \Rightarrow B(?x)) \tag{3.1}$$

serviront à exprimer des significations: par exemple, on a:

$$\forall\ ?\mathtt{R} \quad (\text{Navigable}(?\mathtt{R}) \Rightarrow \text{Voie-de-communication}(?\mathtt{R}))$$

est équivalent à la phase *Toutes les voies navigables sont des voies de communications*. Alors que l'énoncé: *Il y une rivière traversant Paris* s'exprime par:

$$\exists\ ?\mathtt{R} \quad (\text{Rivière}(?\mathtt{R}) \wedge \text{Traverser-Paris}(?\mathtt{R}))$$

Il n'est pas toujours facile de représenter par une formule une connaissance exprimée en langage naturel comme dans l'exemple précédent. Par exemple: *Si deux tronçons sont de la même route, alors ils ont les mêmes propriétés*: on peut proposer **Pro**, et **MemeRoute** comme prédicat:

$$\forall \mathtt{Pro},\ \mathtt{T}_1,\ \mathtt{T}_2\ (\text{MemeRoute}(\mathtt{T}_1, \mathtt{T}_2) \Rightarrow (\mathtt{Pro}(\mathtt{T}_1) \Leftrightarrow \mathtt{Pro}(\mathtt{T}_2))) \tag{3.2}$$

Mais cette expression n'est pas du premier ordre car il y a une quantification sur un prédicat **Pro** .

La principale règle d'inférence, utilisée pour rendre compte d'un raisonnement en logique des prédicats, est la spécialisation universelle qui consiste à substituer dans une formule une variable quantifiée universellement par un terme (instanciation de variables), une variable ou une constante. Cette règle est aussi nommée l'élimination du $\forall$.

La logique, en I.A., s'attache à l'utilisation de la connaissance, et non à sa représentation. Elle trouve tout son intérêt dans une représentation de connaissances déclarative, en particulier pour les règles de production.

Une règle de production est la représentation d'un procédé pour, à partir des formules de connaissances (de type tel que l'expression 3.1), déduire d'autres connaissances. On espère obtenir une connaissance proposée à priori en partant d'un

ensemble fixe de règles d'inférence par application de ces règles un nombre fini de fois.

A l'heure actuelle, les règles de production connaissent un réel succès en tant que moyen de représentation des connaissances dans les systèmes experts.

Les règles de production ne sont qu'une représentation superficielle d'un phénomène plus profond, elles ne cherchent pas à le comprendre mais à le simuler simplement, il faut que leurs bases soient déjà dotées de mécanismes plus ou moins riches. Notre position est donc de n'utiliser les règles de production qu'en dernier ressort, lorsque les autres représentations sont inadéquates.

De plus en plus de programmes appliquent à un système de production des données complexes: schémas, réseaux sémantiques, objets, etc.

Chapitre 4

Les langages orientés objets

La complexité et la taille des projets, comme la généralisation cartographique, aussi bien en moyen humain qu'en matériel, nécessitent des outils de programmation nombreux et perfectionnés pour faciliter la mise en oeuvre et mener les applications à leur terme dans les meilleures conditions. Il est souhaitable que la plus grande part possible du travail soit pris en charge par la machine, afin que les programmes soient faciles à tester, à améliorer, à réutiliser et à maintenir. Ces objectifs sont encore loin d'être atteints à cause de contraintes techniques diverses.

La taille des programmes et les problèmes de maintenance ont mis en évidence la nécessité de promouvoir l'abstraction et la réutilisabilité des logiciels.

On s'est déjà posé la question: *comment sont structurées les connaissances qu'un humain a acquis au cours de ses activités.* Les psychologues ont montré que les êtres humains ont tendance à identifier une famille d'objets et à mener des raisonnements sur ses membres en faisant référence à un objet précis, **typique** de la famille. En partant d'idée que la mémoire humaine dispose de structures d'information bien organisées dont le déclenchement est lié à certaines situations, les chercheurs ont introduit les notions voisines de **schéma**, **frame**, **prototype**, **objet structuré** pour définir des structures algorithmiques complexes de traitement de sens en comparant avec le mode de fonctionnement du raisonnement humain.

La programmation orientée objets est plus un style de programmation qu'un ensemble de langages. Les **objets**, ou **schémas** sont la transposition informatique de la vie réelle: par exemple, sur la figure 4.1 décrit un objet d'une route avec son **nom**, son **code** administratif, sa largeur, sa liste de **noeuds**, sa liste d'**arcs**, ses **voisins** routes, son **type** de voies, son **signe** graphique conventionnel, ses noeuds de **croisements** importants et les connaissances associées (les procédures: comment se changent ses signes conventionnels selon l'échelle, comment déterminer ses voisins actuels, etc).

Les langages à objets ont brisé la dichotomie classique entre données et programmes. On y groupe ensemble les connaissances liées à un même concept de la même manière qu'une recette de cuisine comprend la liste des ingrédients et méthodes de préparation.

Ce style de programmation rend les logiciels extensibles et facilement réutilisables. Un programme n'a pas besoin de savoir de quel type est un objet et quelle est sa représentation interne. Il l'utilise en connaissant simplement ses fonctionnalités. Les

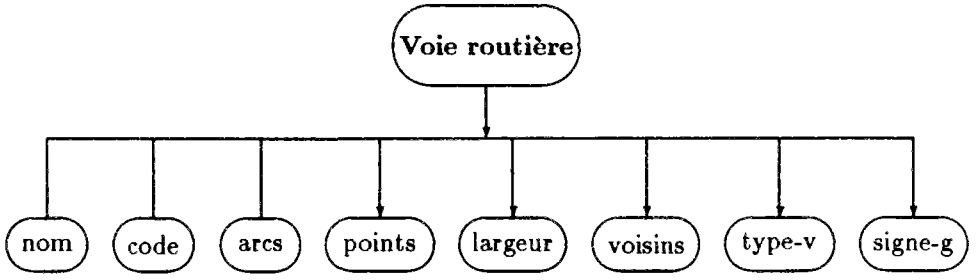


Figure 4.1: Les champs de l'objet de **voie routière**

objets sont des boites noires utilisables dans des contextes divers. Certains n'hésitent pas à les comparer avec des composants électroniques, à des puces.

Les systèmes classiques sont constitués de données et de procédures traitants ces données. Dans les langages orientés objets, il n'existe plus qu'un type de donnée: **OBJET**.

Un schéma se définit comme une structure de donnée idéale, ou prototype, servant de modèle avec d'autres objets à analyser. Un objet particulier est une instance du prototype réalisé.

Un objet représente à la fois un savoir déclaratif et un savoir procédural:

- *savoir déclaratif*: base de données locale: les variables ou champs.
- *savoir procédural*: programme: les méthodes, les procédures de chaque objet.

Un logiciel est alors une collection d'objets différents. La connaissance (données et programmes) est distribuée entre chacune de ces entités.

La programmation objet amène à structurer l'univers d'une application en terme d'objets plutôt qu'en terme de procédures. Trois grandes familles en ajoutant le pionnier les réseaux sémantiques apparaissent naturellement, chacun privilégiant un point de vue sur la notion d'objet.

- Les *réseaux sémantiques* sont les premiers systèmes à héritages ayant été utilisés en représentation des connaissances. Ils ont eu une influence primordiale sur tous les langages de représentation à base d'objets, en particulier sur les langages à frames.
- *langage à classes*: la classe est un type de données, qui définit un modèle pour la structure de ses représentants physiques et un ensemble d'opérations applicable à cette structure.
- *langage à frames*: le frame est une entité de connaissance représentant le prototype d'un concept, les autres objets ressemblant à ce prototype prennent référence sur lui. C'est la catégorie du langage de frame.
- *langage à acteurs*: l'acteur est une entité autonome et active, qui se reproduit par copie, il privilégie la notion de comportement individuel.

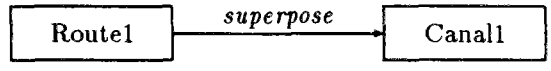
De nombreux langages intègrent des caractéristiques empruntées à une et à l'autre. Ils constituent une 5ieme famille, appelés langages hybrides.

4.1 Les réseaux sémantiques

Un réseau sémantique est un ensemble de noeuds *typés* appelés concepts représentant les concepts de l'univers de référence, reliés par des arcs *orientés* étiquetés exprimant les relations qui peuvent exister entre ces noeuds. Les concepts peuvent être des entités, des individus, des situations, etc. Un ensemble d'opérations associé au réseau permet de l'exploiter: création et suppression de noeuds et d'arcs, parcours du graphe des relations, mécanismes d'héritages et de filtrage, etc.

Les noeuds jouent le rôle des termes et les arcs jouent le rôle des prédicats en logique.

Un réseau sémantique décrivant une phrase se représente naturellement par un graphe. Par exemple, le fait *la route*$_1$ *se superpose au canal*$_1$ est représenté par la figure suivante:



L'arc orienté représente l'action dont l'acteur est le noeud de départ et l'objet le noeud d'arrivée. Sous une forme non graphique, on pourra avoir un triplet:

(**superpose Route1 Canal1**).

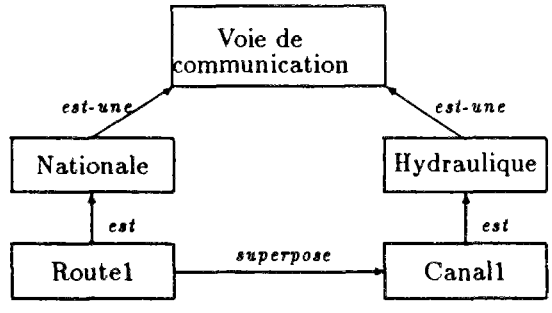


Figure 4.2: Réseau communication

En général, les concepts sont reliés à la famille à laquel ils appartiennent.
Les avantages d'une telle représentation sont multiples:

- un élément n'apparaît qu'une seule fois, alors qu'il peut intervenir dans la description de plusieurs concepts;
- de nombreuses relations entre les éléments ne sont pas exprimées, l'interpréteur du réseau les retrouve: par exemple la transivité de la relation **est** permet de décrire des hiérarchies entre les concepts. et on parle d'héritages des propriétés. Les assertions affirmées dans un noeud supérieur sont également vraies dans les noeuds inférieurs. On peut alors spécifier des valeurs par défaut.

  Dans la figure 4.2, on peut déduire que la route$_1$ et la canal$_1$ sont de la classe de voie de communication en remontant le lien **est**. La recherche d'une propriété

dans la hiérarchie d'héritage est assimilée à une inférence, appelée *inférence par héritage.*

Si la représentation d'une relation binaire est un arc, celle d'une relation n-aire est un noeud, les informations associées à une action décrite dans le réseau sont des relations attachées au noeud figurant l'action. Par exemple, la description *les objets se superposent* peut être décrite précisément en indiquant l'objet du dessus, l'objet du dessous, le lieu, la longueur de superposition; elle était simplement figurée par l'arc **superpose** dans la figure 4.2, on verra que l'action **superposer** pourra très bien être représentée par un noeud de même nom dans le réseau de la figure 4.4.

Les exemples précédents n'ont montré que des réseaux bien spécifiques, où les arcs qui relient les différents noeuds dépendent étroitement des phrases représentées. Pour modéliser des connaissances plus générales sous forme de réseaux sémantiques, il faut introduire des liens structurels exprimant les propriétés intrinsèques des concepts associés aux noeuds, on distingue alors ATPG *attribution de propriétés génériques* et ATPS *attribution de propriétés spécifiques* (figure 4.3) dans [15].

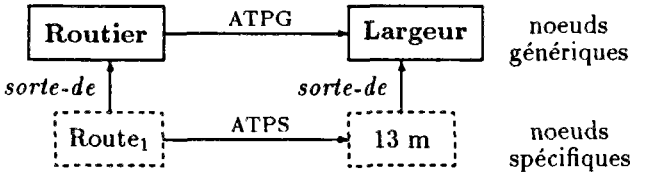


Figure 4.3: Le lien ATPG et le lien ATPS

Ces liens permettent de structurer un réseau en une partie spécifique, décrivant les données particulières à la phrase traitée, et une partie générique, réutilisable. La première est en quelque sorte une *instanciation* de la seconde.

Il faut d'ailleurs définir avec précision les concepts et leurs inter-relations. Ainsi la phrase *un tronçon de la route$_1$ et un tronçon du canal$_1$ se superposent à partir de l'échelle 1:250 000* ne pourra trouver de représentation adéquate qu'en augmentant le réseau précédent: on passera à une structure dans laquelle le concept **superposer** est l'élément principal du réseau. Cette solution consiste à permettre à des noeuds de modéliser des situations, des actions, aussi bien que des objets.

La puissance d'un formalisme de représentation provient en grande partie des méthodes d'inférence qu'il propose pour résoudre des problèmes. Deux méthodes sont couramment utilisées dans les réseaux sémantiques, l'inférence par héritage et le filtrage, qui permet de rechercher des informations dans le réseau par accès associatif.

Pour raisonner, il consiste à représenter par un réseau le contenu du but cherché, qui est ensuite mis en correspondance avec la base de données du réseau. Une demande d'informations est modélisée par un réseau où les noeuds représentant les informations inconnues sont étiquetés par des variables. Le rôle de la procédure de filtrage consiste à mettre en correspondance noeud à noeud ce réseau et un sous-ensemble du réseau représentant la connaissance disponible, afin de déduire les valeurs des variables. Autrement dit: le fragment de réseau issu du but cherché contient des noeuds variables qui, suivant le principe de l'unification, prendront des valeurs de façon à obtenir une correspondance parfaite entre ce réseau et la partie concernée dans le réseau de la base de données. Ainsi la question *quelle route est superposée avec le*

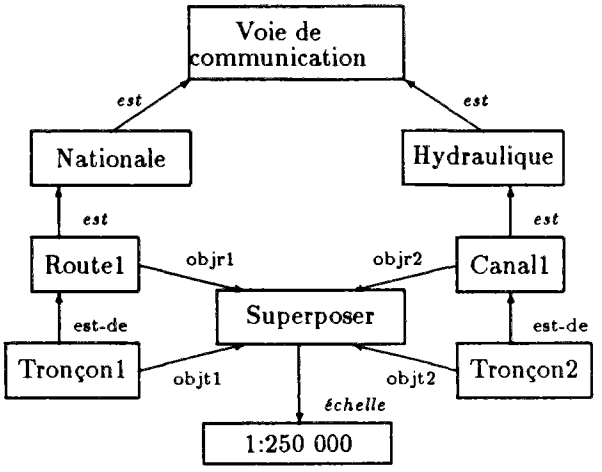
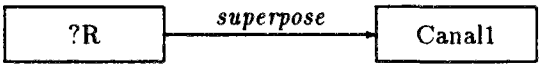


Figure 4.4: Réseau à l'echelle E= 1:250 000

canal$_1$ *?* sera résolue, avec le réseau sémantique donné à la figure suivante en décrivant à l'interprète le graphe:



Un propre algorithme de filtrage associé donnera rapidement une réponse souhaitée: ?R = Route1. Il peut y avoir plusieurs réponses dans d'autres cas.

On peut décrire sur ce modèle des situations plus complexes et non explicitement représentées dans le réseau sémantique, mais la formulation par un programme ou un expert d'un réseau sémantique de plusieurs milliers de noeuds est une tâche très difficile, on doit utiliser des heuristiques pour sélectionner les éléments du réseau qui doivent être utilisés en priorité, en évitant ainsi, lorsque le réseau devient trop important, des problèmes combinatoires.

On s'est orienté plutôt vers la génération automatique de tels réseaux à partir de règles.

Il existe des modèles sémantiques des bases de données qui s'affranchissent de la première forme normale du modèle relationnel. Elles proposent les mêmes outils de structuration de l'information que les langages à objets: par exemple, le principe d'abstraction de généralisation/spécialisation est analogue à l'héritage.

Cependant, les modèles sémantiques sont avant tout structurels, dans la mesure où ils permettent de définir la *structure* des entités apparaissant dans une application, alors que l'approche objet favorise la modélisation du *comportement* des entités. Or, la notion de comportement, souvent appelée composante dynamique dans les bases de données, s'avère extrêmement importante pour les bases de données de dessin assistés par ordinateur. Malheureusement, les langages associés aux modèles sémantiques sont avant tout des langages d'interrogation et les langages de manipulation qui vont de pair ne respectent pas le principe d'encapsulation, pourtant essentiel pour développer des applications importantes.

Les modèles objets se manifestent bien dans les applications à forte composante dynamique. Il reste cependant beaucoup de problèmes à résoudre pour les adapter à

la problématique des bases de données: gestion de la persistance des objets au-delà de la simple exécution d'un programme, accès concurrents aux objets dans un contexte multi-utilisateurs, évolution dynamique du schéma de la base ou du comportement des objets eux-mêmes.

4.2 Les prototypes

Un concept décrivant une catégorie d'individus peut donc être matérialisé par l'un d'entre eux, généralement le plus connu, appelé prototype. Il est considéré comme un membre *moyen*, représentant l'idéal de la catégorie. Un prototype possède les caractéristiques les plus fréquemment rencontrées chez les membres de la catégorie: il représente une connaissance par défaut qui n'est prise en compte qu'en l'absence de toute autre information.

Un individu n'est pas moulé sur le modèle d'une classe et ne contient les seuls comportements et caractéristiques qui diffèrent de ceux du prototype.

Par exemple, on définit un prototype *autoroute* par des caractéristiques suivantes: *à chaussées séparées, voies larges, vitesse grande*; ces caractéristiques étant les plus fréquemment rencontrées chez tous les membres de la catégorie *autoroute*. Pour en définir un membre *autoroute-privée*, il possédera les seuls comportements: *propriété privée, payant*, qui diffèrent de ceux de son prototype *autoroute*.

Pour retrouver la description complète d'une autoroute-privée, il faut donc examiner d'abord ses propres descriptions: *propriété privée, payant*, puis celles du prototype *autoroute*: *à chaussées séparées, voies larges, vitesse grande*.

4.3 Les classes et objets

Les langages de classes distinguent deux concepts: les classes et les instances, ou les schémas et les objet particuliers. La classe s'apparente à un type abstrait de données et décrit un ensemble d'objets partageant la même structure et le même comportement. Elle sert de moule pour générer ses représentants physiques, ou instances.

Un objet est donc un module élémentaire réunissant un certain nombre de données qui lui sont propres et des procédures qui les manipulent. On peut le voir comme un serveur capable d'effectuer des requêtes sur sa propre structure, d'en masquer une partie ou d'en permettre un accès contrôlé. C'est une portion de connaissance possédant son propre contrôle et pouvant vivre indépendamment de ses congénères.

Au sens LISP, un objet peut être vu comme un environnement, c'est à dire une collection de liaison (**symbole** $\leftrightarrow$ **valeur**), et (**symbole** $\leftrightarrow$ **fonction**). Deux objets peuvent utiliser les mêmes symboles en y associant des valeurs différentes.

La représentation de la structure d'un objet s'exprime en 2 niveaux:

1. **abstrait**: tout objet générique est représenté par un modèle appelé **Classe** qui décrit la structure et ses propriétés. C'est une représentation d'un type abstrait, un moule à partir du quel on fabrique autant d'exemplaires que l'on veut. Par exemple, pour décrire un modèle de toutes les autoroutes en France, on peut définir une classe **Autoroute** dotée des champs analogues à ceux de la

Voie routière dans la figure 4.1 plus éventuellement un champ supplémentaire comme *vitesse* décrivant la vitesse limitée maximum pour la sécurité.

On peut également définir une classe **Chemin** pour décrire les routes moins importantes.

L'ensemble des classes constitue la base de classes.

2. **concret**: tout objet spécifique est une **instance** de la classe auquel il appartient, les instances de la même classe ont des propriétés communes qui sont définies par leur classe. La création d'une instance est appelée *instanciation*. On peut donc se servir du moule de **Autoroute** et **Chemin** qu'on a défini ci-dessus pour *fabriquer* des autoroutes et des chemins réels en donnant les valeurs exactes de leur largeurs, leur liste des arcs, des tronçons, des signes conventionnels, etc. L'ensemble des instances forme la Base des données du domaine d'application.

Une classe décrit des informations qui sont complétées et enrichies pour définir d'autres familles d'objets. Les informations les plus générales sont ainsi mises en commun dans des classes à partir desquelles sont créées des sous-classes successives, contenant des informations de plus en plus spécialisées, l'ensemble des classes forme une hiérarchie dans laquelle chaque sous-classe hérite des données et des méthodes de ses classes mères.

Une classe spécifie essentiellement la structure de données sur laquelle est bâtie chaque instance par la liste des champs et des méthodes. Cette spécification peut être plus précise, par exemple avec les valeurs attribuées par défaut aux champs ou la définition des facettes.

Par exemple, on peut définir sur la figure 4.1 une classe **Carrefour** avec

- une structure de données, ou des champs, constituées des variables d'instances:
  - nom,
  - sommet,
  - liste d'arcs,
  - les carrefours voisins proches,
  - les carrefours reliés directement,
  - etc.
- une listes des méthodes, ou des procédures pour effectuer des requêtes sur sa propre structure et définir ses relations sémantiques avec d'autres objets:
  - **dwd**: calculer les projections de la largeur entre les arcs voisins,
  - **Routes-principales**: savoir les routes principales qui passent à ce carrefours et leurs types, les 2 routes les plus importantes de ce carrefour,
  - **angles-arcs**: les angles entre les arcs voisins,
  - **domaine-convexité**: savoir s'il a un domaine de convexité[1] non vide, et un sommet de convexité non nul,

[1] Défini dans la section 6.6.2

- **Dis12sc**: la distance avec un voisin quelconque en déduisant les projections des effets des largeurs des routes, et si elle est inférieure au seuil de séparation[2].
- **Voisins-dangers**: actuellement, tous les voisins trop proches autour,
- **S1>S2?**: s'il est prioritaire par rapport à un autre carrefour,
- **Où-name**: la meilleur position pour poser un nom de toponyme,
- **Confondre**: comment déménager chez un voisin si nécessaire,
- **Delete**: comment mourir en cas de suppression de ce carrefour, c'est à dire arranger les affaires de testament, d'héritages entre les arcs qui passent à ce point.
- **Direction-d**: les meilleures directions de déplacement en cas de besoin.
- etc.

Nous pouvons ensuite modéliser tous les carrefours qui connaîtront leur propre structure, leurs propriétés, qui sauront se déplacer comme il faut. Une classe représente donc un ensemble dont les éléments sont les instances.

Lorsque chaque classe ne comporte que des couples <attribut, valeur>, la base de données résultante est équivalente à un réseau sémantique.

4.4 Les frames

Héritiers des réseaux sémantiques, généralement, le modèle de représentation des frames ne se réduit pas à un ensemble de couples <attribut, valeur>. Un frame est un prototype décrivant une situation ou un objet standard, l'ensemble des frames est organisé en une hiérarchie d'héritage où, à la différence des classes, tout objet est à la fois un représentant des frames dont il est issu et un générateur de frames plus spécialisés.

Un frame possède des attributs. Chaque attribut peut être caractérisé par un ensemble de facettes, déclaratives ou procédurales, ou aspects sémantiques, qui donne aux schémas une structure de réseau sémantique généralisé lorsque les éléments à manipuler sont complexes.

Les facettes procédurales correspondent à des descripteurs qui indiquent les procédures (démons) à activer en cas d'accès ou de modification de cet attribut. On peut avoir deux types de facettes:

facette déclarative : elle permet de spécifier des valeurs par défaut, c'est à dire qu'en cas d'information incomplète, on peut donner une valeur à un attribut. Ce type de facette permet aussi de décrire un ensemble de valeurs possibles pour un attribut.

facette procédurale ou attachement procédural: elle a un caractère actif et est exécutée lors de l'acquisition d'une valeur pour l'attribut. on peut distinguer les types d'attachement procéduraux:

- ▸ *si-besoin*: exécuté lorsque la demande de la valeur de l'attribut est effectuée;

[2]défini dans la section 2.4

- *si-ajout*: exécuté lorsque l'attribut reçoit une valeur;
- *si-suppression*: exécuté lorsque la valeur de l'attribut est enlevée;
- *si-lecture*: exécuté à chaque lecture de la valeur de l'attribut.

Un frame n'a donc pas de comportement propre décrit par des méthodes: les opérations de manipulation des attributs sont détenues par les attributs eux-mêmes. En général, il est distingué des autres par le fait que son unité de base est le triplet: (**frame, attribut, facette**).

4.5 Les acteurs

Un acteur est un objet actif qui communique avec ses semblables par envois de messages asynchrones. Il connaît des acteurs, appelés ses *accointances*, qui jouent le rôle des variables d'instance. Son comportement est défini par un *script* qui décrit la manière dont il réagit aux événements provoqués par les autres acteurs. Contrairement aux classes, les acteurs ne sont pas organisés hiérarchiquement: chaque acteur est en relation avec d'autres acteurs auxquels il **délègue** les messages qu'il ne peut pas traiter, réalisant ainsi l'équivalent du mécanisme d'héritage.

Un acteur peut en créer un autre par copie de lui-même. La copie engendrée est soit rigoureusement identique, soit différentielle, c'est à dire spécialisée par adjonction de nouvelles caractéristiques. Ce mécanisme ressemble à l'instanciation des langages de classes.

Par son coté dynamique, le modèle proposé par les langages d'acteurs se prête particulièrement bien à la programmation parallèle et à la mise en oeuvre de systèmes répartis sur un réseau de processeurs.

4.6 Les langages hybrides

Typiquement, un langage hybride permet d'encapsuler des méthodes dans un frame ou bien d'associer des facettes aux variables d'instance d'une classe, ce qui permet de bénéficier conjointement des avantages offerts par les classes et les frames. La plupart du temps, il comprend également une composante de programmation logique.

4.7 Les messages

Les objets communiquent entre eux par l'envoi et la réception de messages. On a accès aux objets ou on les utilise par un moyen unique: l'envoi de *message*. L'envoi de message repose sur le concept de **type abstrait**. Ce sont des types d'objets dont l'implantation est cachée aux programmes utilisateurs. La seule chose utile aux utilisateurs d'un objet est la documentation de ses opérations spécifiques. Un langage supporte les types abstraits lorsqu'il permet de regrouper sous un même type un ensemble de procédures qui lui sont spécifiques.

Vis à vis de la programmation fonctionnelle, l'envoi de message précise le destinataire sans se soucier de son type. Le même message peut être interprété de manière différente par des destinataires de types distincts.

Puisque chaque objet contient son propre mode d'emploi, un même message envoyé à divers objets pourra donner des résultats différents.

Dans cette optique, une grande partie des problèmes de syntaxe disparaît puisque toutes les communications ont la même forme, seuls les objets receveurs permettent de les différencier.

C'est, en effet, le destinataire du message-ou sa classe- qui possède le code du programme à exécuter. Par exemple, on a déjà défini deux types de classes: **Voies routières** et **carrefour**, on définit une méthode pour **se déplacer**. Pour bouger un des objets de ces deux classes, il suffit de lui envoyer le message: **déplacer**. En fonction de son type, chacun agira selon son mode particulier de déplacement.

- un carrefour va consulter ses environs, et essayer de bouger son point du sommet dans son domaine de convexité[3] si c'est possible, et donne des informations en cas d'impossibilité de rester dans ce domaine.
- une route va d'abord envoyer des messages correspondants à tous ses noeuds qui vont réagir en fonction de leur type et environnement, et ensuite corriger les défauts de changement de convexité de l'ensemble de la route[4] en linéarisant les petites sinuosités.

Un message tel que (**écarter R5 dir dis**) spécifie en général:

- le nom destinataire, comme le nom de la route: **R5**
- un sélecteur: un nom de fonction en quelque sorte, par exemple, l'action **écarter**.
- des arguments éventuels, la direction *dir* et la distance *dis* pour l'action **écarter** une route.

Dans les langages objets, programmer revient à définir un ensemble d'entités actives, spécialisées, et une structure dynamique: les méthodes, qui définissent le mode d'emploi de l'objet.

L'ensemble des méthodes constitue l'interface par laquelle un objet interagit avec l'extérieur. L'accès et la modification des champs peuvent se faire également par envoi de messages.

4.8 L'héritage

Il s'agit d'un problème de partage de connaissances. La classe doit être considérée comme un réservoir de connaissances à partir duquel il est possible de définir d'autres classes plus spécifiques, complétant les connaissances de leur class mère. Les connaissances les plus générales sont ainsi mises en commun dans des classes qui sont ensuite *spécialisées* par définitions de *sous-classe* successives, contenant des connaissances de plus en plus spécifiques. Une sous-classe est donc une spécialisation de la description d'une classe, appelée sa *sur-classe*, dont elle partage les variables et les méthodes.

La figure 4.5 décrit les sous-classes de la classe **Réseau de communication**, c'est un graphe réduit de la figure 7.1 de la page 7.1.

[3] défini dans la section 6.6.2

[4] défini dans la section 6.4.2

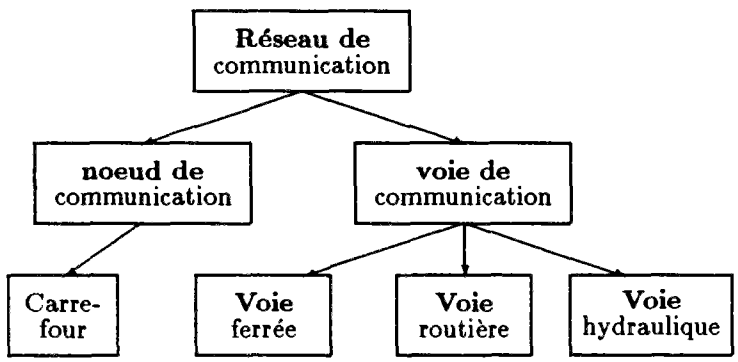


Figure 4.5: Les sous-classes de la classe de `réseau de communication`

Bien sûr, au niveau physique, les duplications inutiles sont évitées, mais conceptuellement tout se passe *comme* les informations de la sur-classe était recopiées dans la sous-classe: une sous-classe *hérite* des informations de sa surclasse. La spécialisation d'une classe peut être réalisée selon deux techniques:

- *enrichissement*:

  Une classe peut être définie comme sous-classe d'une autre, les instances de la sous-classe héritent alors automatiquement des propriétés de la sur-classe, la sous-classe *spécialise* sa sur-classe par addition et substitution de variables ou de méthodes.

  Par exemple, on crée une sous-classe **passage-à-niveau** de la classe **carrefour**, cette classe a la propriété que tous les arcs arrivant à ce point sont tous au même niveau, elle a ses méthodes propres comme signe graphique et variables propres et tout passage-à-niveau saura se déplacer et aura un nom, un sommet, des arcs,...etc. comme un **carrefour** normal.

  La sous-classe peut être dotée de nouvelles variables et/ou de nouvelle méthodes représentant les caractéristiques propres au sous-ensemble d'objets ainsi décrit.

  Par exemple: On définit une classe **Route** avec six variables d'instance: **nom largeur lvoisins axe larcs lnoeuds** décrivant des routes principales sans préciser le nombre de voies en supposant 2 par défaut, une nouvelle variable **nombre-de-voies** est ajouté à la classe plus spécifique de routes qui ont plus de 3 voies: **{R+3voies}**. par héritage, la classe **{R+3voies}** compte en fait sept variables d'instance:

  nom largeur lvoisins axe larcs lnoeuds nombre-de-voies

  Une classe est donc définie par ses propres champs et des méthodes plus ceux qui sont hérités de ses surclasses.

- *substitution* ou *masquer*. Eventuellement, une méthode ou une variable peut être redéfinie dans une sous-classe et donc remplacer celle qui aurait dû être héritée. on peut donner une nouvelle définition à une méthode héritée, lorsque celle-ci se révèle inadéquate pour l'ensemble des objets décrits par la sur-classe.

Par exemple: on définit une méthode **éliminer** générale au niveau de la classe **voie routière** pour la suppression d'une route en fonction de la taille de la route selon les règles de lisibilité, c'est une règle quantitative. selon la règle qualitative[5], on redéfinit la méthode **éliminer** au niveau inférieur de la classe **voie routière**, par exemple à la classe **sentier**, qui doit être éliminée à partir de l'échelle 1:100 000 selon son importance.

L'héritage permet de définir des classes d'objets qui sont semblables à d'autres permettant une structuration hiérarchique de la connaissance. Le concept d'héritage existe dans tous les langages objets.

L'héritage permet de factoriser des connaissances pour créer de nouveaux objets, par spécialisation ou extension des objets préexistants. Il repose sur deux composantes:

- *statique*: c'est la relation qui lie un objet à celui ou ceux dont il hérite. Elle est représentée par le graphe d'héritage et possède une sémantique.
- *dynamique*: c'est le processus qui réalise l'héritage, en donnant aux objets l'accès aux informations dont ils héritent.

La relation d'héritage est *transitive*: les caractéristiques des classes supérieures sont héritées par les classes inférieures, qui sont d'autant plus spécialisées qu'elles sont proches des feuilles de l'arbre du graphe d'héritage.

La relation *classe-instance* est assimilable à un lien d'appartenance: *élément-ensemble*, la route *A4* de type *autoroute* est une des routes qui appartient à l'ensemble de *voies routières*. L'héritage doit être considéré comme représentant une inclusion entre un ensemble et un sous-ensemble, la route *A4* de type *autoroute* a un nom, des caractéristiques de voies, de largeur comme toutes *voies routières*.

L'héritage peut être simple ou multiple pour des objets composites tels qu'une route, à la fois voie de communication et éventuellement aussi limite administrative. Dans le second cas, la sous-classe hérite de plusieurs classes, elle comprend alors l'union des variables et des méthodes de toutes ses sur-classes. Des conflits peuvent alors se déclarer entre les informations héritées de différentes surclasses. Il faut donc définir un moyen de les résoudre. La solution la plus simple est de donner une liste de priorité de classes qui détermine de quelle classe on hérite les variables ou les méthodes en conflit[6].

La structuration en classe et sous-classe entraîne une *modularité* importante: la description de l'univers est divisée en partie indépendantes regroupant les objets par affinité. La modification de l'une d'elles n'entraîne qu'un minimum de modification des autres. Ce fait est bien mis en évidence par le graphe d'héritage: chaque sous-arbre regroupe des objets partageant les caractéristiques de sa racine. En principe, la modification de cette racine n'a d'incidence que sur ses sous-classes.

L'héritages permet la réutilisation des programmes, on peut s'arranger pour profiter de cet avantage: certaines classes ne sont créées que pour en faire des **réservoirs d'héritage** pour les autres classes, par exemple sur figure 4.5 la classe **Voies de communication** est définie pour définir ensuite les sous-classes: **Voies routière**, **Voies ferrées**, etc.

[5] voir la section 6.2

[6] La section 4.11 en discute un en détails

Instancier de telles classes n'a généralement pas de sens et représenterait de toute façon guère d'intérêt. Par exemple, la classe très générale et abstraite: **Voies de communication** n'est pas destinée à être instanciée, elle groupe toutes les méthodes communes à toutes les sous-classes: **voie routière**, **voie hydraulique**, **voie ferrée**, etc. ces dernières hériterons toutes les propriétés de la sur-classe **voie de communication**. Il y a peu d'intérêt à créer des instances de la sur classe **voie de communication** qui ferait créer des voies de communication incomplètes, trop abstraites, par contre, toutes les autres classes sont plus spécifiques qu'elle, mais celles-ci sont aussi abstraites et sont destinées à créer des sous-classes telle que **rivière** de la **voie hydraulique**, **Route** et **Chemin** de la **Voies routières**, etc. (voir le graphe complet des classes et sous-classes du réseau de communication sur la figure 7.1)

Un accent devrait être porté sur le fait que l'évolution des connaissances nécessite la prise en compte d'exception et de modification du graphe d'héritage, pouvant entraîner l'apparition de contradictions et de redondances. La section 4.11 donne quelques exemples.

4.9 La relation d'héritage

La relation d'héritage peut être interprétée de différentes façons selon l'usage qui en fait est.

- *conceptuel*: la relation d'héritage indique une spécialisation. Par exemple, la relation qui lie une espèce animale à ses sous-espèce est de cette nature:

  Une voie routière *est une sorte de* voie de communication

  Les classes les plus générales se trouvent près de la racine du graphe d'héritage tandis que les classes décrivant des objets particuliers se trouvent près des feuilles.

- *ensembliste*: La relation d'héritage décrit une inclusion d'ensemble. La classe {**VoieRoutière**} héritant de la classe {**VoiedeCommunication**}, l'ensemble des éléments (instances) de la classe {**VoieRoutière**} est inclus dans l'ensemble des éléments de la classe {**VoiedeCommunication**}:

  $$(\texttt{VoieRoutière} \subseteq \texttt{VoiedeCommunication})$$

  A ce propos, les langages de classe font une nette distinction entre classes et instances. Si la relation d'héritage peut être assimilée à une inclusion ensembliste ($\subset$) entre deux classes, la relation liant une instance à sa classe se représente comme l'appartenance($\in$) d'un élément à un ensemble. Il est convenu de ne pas parler d'héritage. Cette distinction classes-instances est toutefois souvent remise en cause, en particulier dans les langages dédiés à la représentation des connaissances.

- *logique*: L'appartenance à une classe est exprimée par un prédicat. Si l'élément x appartient à la classe { **VoieRoutière** }, alors **VoieRoutière**(x) est vrai . La formule *une voie routière est une voie de communication* se traduit alors par:

  $$\forall x \quad (\texttt{VoieRoutière}(x) \quad \Rightarrow \quad \texttt{VoiedeCommunication}(x))$$

Toutes les relations correspondant à ces différentes interprétations sont *transitives* et déterminent un *preordre* sur les classes.

Aujourd'hui, les langages orientés objets sont amenés à tenir une place de plus en plus prépondérante dans la conception des logiciels. Pour notre but qui est d'automatiser une partie des opérations de généralisation cartographique, la programmation objet tend à servir de base aussi bien pour la réalisation de moteur d'inférences (cf. la section 8) que pour la construction de bases de connaissances pour systèmes experts (cf. la section 7.1.1) .

Les langages orientés objets sont des langages de programmation à part entière. Néanmoins, ils diffèrent des langages classiques(Pascal, etc.) dans la manière de résoudre les problèmes et de contrôler le déroulement des programmes.

La programmation orientée modifie radicalement notre vision traditionnelle de programmeur (définir des variables et des procédures qui sont activées dans un ordre prévu par la séquence des instructions du programme).

Les difficultés sont en fait, dans une application, de déterminer quelles sont les sortes d'objets à créer et quels messages ceux-ci devront supporter afin de permettre un dialogue efficace. D'une manière générale, on cherche à définir des objets représentatifs du monde réel, comportant peu de variables et de méthodes et ayant une forte cohérence fonctionnelle ou organique. On minise ainsi les interactions entre objets, cela se traduit par une bonne intelligibilité et une plus grande facilité de mise au point.

La section 7.1.1 discute les constructions et structurations des objets.

4.10 Objet et représentation des connaissance

Les objets sont bien adaptés aux méthodes de descriptions incrémentales, où les connaissances spécifiques sont obtenues par affinement progressif de connaissances génériques. Le mécanisme d'héritage offre une meilleur solution au niveau de partage de connaissance. Ils permettent donc de traiter plus facilement certains des problèmes fondamentaux dans la représentation de connaissance:

- **Typicalité, connaissance par défaut et exception**
  Par définition, un objet ou un prototype représente des connaissances typiques, des informations par défaut ou implicites caractérisant une famille d'individus. C'est une référence, utilisée pour comparer des objets qui doivent être reconnus, analysés ou classés. Les propriétés exceptionnelles peuvent être masquées[7] chez un individu sans perturber la hiérarchie déjà établie.

- **Modification de l'univers et informations incomplètes**
  A l'origine incomplète, la connaissance décrite par un prototype s'affine au fur à mesure que la connaissance sur les objets représentés augmente. Puisque l'héritage est dynamique, toute modification dans un prototype est implicitement répercutée sur les objets qui dépendent du prototype, ces derniers ne possédant que virtuellement les propriétés qu'ils partagent avec lui.

[7] une propriété peut être *masquée* veut dire qu'elle peut être redéfinie

4.11 Les exceptions

Le monde n'est pas parfait, les exceptions sont partout. Elles sont tellement courantes qu'il serait anormal qu'il n'y ait pas d'exception dans des domaine tels que le traitement des informations géographiques. Le traitement des exceptions constitue un des problèmes les plus difficiles de la représentation des connaissances et son études théorique est encore en pleine évolution[15].

4.11.1 Héritage et exception

Dans un système où l'héritage est le mécanisme du partage de connaissance, les propriétés d'un concept et leurs valeurs sont *obligatoirement* héritées par les spécialisations du concept. En général, l'héritage des variables d'instance est statique et celui des méthodes est dynamique. Cependant, il est difficile, voire impossible, de construire une abstraction ne décrivant que des propriétés valides pour toutes ses spécialisations. Certaines d'entre elles peuvent représenter des individus exceptionnels, pour lesquels il est nécessaire de masquer les valeurs par défaut de certaines des propriétés héritées. Par exemple, si le concept d'**Arc** est ajouté dans une hiérarchie décrivant les **Voies Routières**, la propriété **List-arcs**, supposons définie avec la valeur de domaine **liste** pour les voies routières, est redéfinie avec la valeur du domaine **atom** ou **nil** pour le concept d'**Arc**, puisqu'un arc ne contient que lui même.

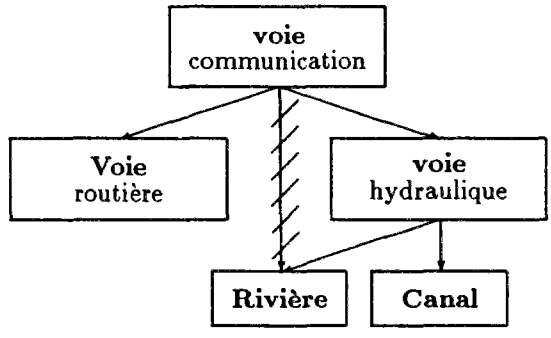


Figure 4.6: Graphe d'héritage, exception au niveau de **Rivière**

Dans la figure 4.6, on définit une sous-classe **Rivière** de la sur-classe **Voie hydraulique** qui est aussi une sous-classe de la classe **Voie de communication**, cette dernière ayant des variables d'instances:

nom largeur lvoisins axe larcs lnoeuds

et une liste de méthodes pour des processus de traitement. Donc une **Rivière** est défini en tant que **Voie hydraulique** qui est lui même une **voie de communication**. En général, une rivière n'est pas comme une voie ferrée, ni comme les routes qui ont une largeur régulière et constante, donc la variable d'instance **largeur** et **axe** ne peuvent pas décrire une rivière non régulière. Par conséquent, les méthodes concernant la largeur et l'axe d'une voie de communication devraient être redéfinies. Plutôt que de masquer dans **Rivière** toutes les propriétés de **voies de communication**,

il est préférable de créer un *arc exceptionnel*, ou *lien d'exception*, entre **voies de communication** et **Rivière**, pour indiquer que le second objet n'hérite pas de caractéristiques du premier.

Donc, chaque fois qu'une propriété serait ajoutée ou modifiée dans **voie de communication**, il faudrait vérifier sa compatibilité avec **Rivière**. Le maintien de cohérence du graphe d'héritage serait très difficile à assurer et ne pourrait pas être automatisé, faute d'indication *explicite* sur les incompatibilités existant entre les concepts liés hiérarchiquement.

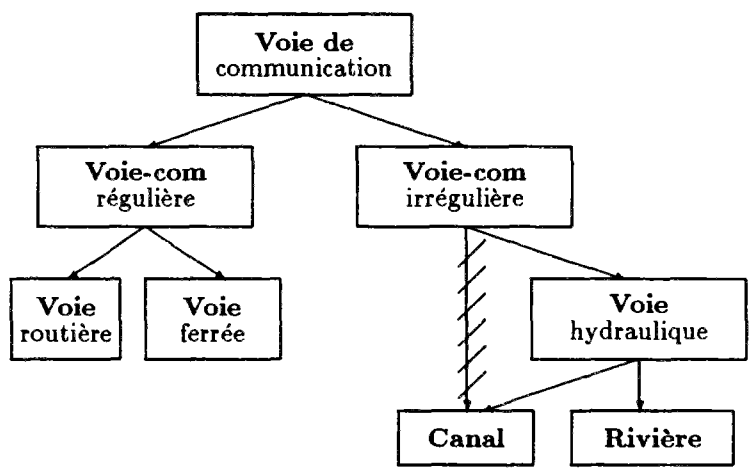


Figure 4.7: Une exception au niveau de **Canal**

Les exceptions font perdre sa transitivité à la relation d'héritage: une rivière n'est pas voie de communication qu'on a spécifié comme modèle, avec *régulière* comme caractéristique, si une rivière n'est pas une voie de communication, cette exception porte-t-elle sur le seul objet excepté ou bien sur tous les ascendants de ce dernier?

Ou bien faut-il encore ajouter d'autres concepts plus spécifiques au dessous ou en dessus du concept **voie de communication**? Ce n'est pas impossible, seulement d'autres exceptions pourraient bien apparaître. Par exemple, deux nouvelles sous-classes: **Voie de communication régulière** et **Voie de communication irrégulière**, et dans ce cas, les **Voie routière** et **Voie ferrée** seront des sous-classes de la première, et **Voie hydraulique** sera de la second. Mais si la classe **Canal** était une sous-classe de la sur-classe **Voie hydraulique**, alors elle n'est pas du tout de caractéristique **irrégulière** comme sa sur-classe ou comme son camarade **Rivière**, donc une autre exception se révèle. Ce qui justifie la fréquence des exceptions.

Ce problème doit être réglé pour pouvoir ajuster en conséquence les algorithmes d'héritage.

4.11.2 Chemins d'héritabilité

La liste de priorité d'un objet détermine l'ensemble des objets dont il hérite, appelé ensemble d'héritabilité ou ensemble des objets héritables. Touretzky[19], [15] exprime l'héritabilité en faisant intervenir la notion de chemin d'héritabilité:

> Un chemin d'héritabilité est un chemin du graphe d'héritage dont aucun arc de transitivité[8] n'est exceptionnel et dont seul le dernier arc peut être exceptionnel. Le chemin est dit **négatif** si le dernier arc est exceptionnel, **positif** sinon.

Par exemple: sur la figure 4.6, entre **Rivière** et **Voie communication**, il existe un chemin d'héritabilité positif: **Voie communication** ← **Voie hydraulique** ← **Rivière**, et un chemin négatif: **Voie communication** ↚ **Rivière**. Le chemin **Voie communication** ← **Voie hydraulique** ← **Rivière** n'est pas un chemin d'héritabilité, car il existe un arc de transitivité exceptionnel: **Voie communication** ↚ **Rivière**.

L'héritabilité d'un objet se définit alors en fonction des chemins d'héritabilité issus de cet objet:

> Un objet o2 est *héritable* par un objet o1 s'il existe un chemin d'héritabilité positif entre o1 et o2. L'objet o2 n'est pas héritable s'il existe un chemin d'héritabilité négatif entre o1 et o2.

L'existence simultanée de chemins d'héritabilité positif et négatif entre deux objets introduit une *contradiction*, puisqu'un objet comme **Voie communication** est à la fois héritable et non héritable par un sous-objet: **Rivière**.

Une contradiction est une *redondance contradictoire* si l'un des chemins d'héritabilité passe par un arc de transitivité de la relation d'héritage. La redondance est facile à reconnaître et à traiter. Il suffit de supprimer un arc redondant pour lever la contradiction.

Une contradiction qui n'est pas due à un arc de transitivité est dite *pure*. Il est nécessaire d'introduire des critères supplémentaires pour trancher.

4.11.3 Quelques techniques de résolution

Le plus court chemin

La technique du *plus court chemin*, utilisée par Scott E. Fahlman, a été la première solution proposée pour traiter les exceptions dans un graphe d'héritage.

Elle consiste à construire l'ensemble d'héritabilité en ne conservant que les objets accessibles par les plus courts chemins d'héritabilité positifs. En cas de contradiction sur une propriété, celle-ci est héritée si le chemin positif est *strictement* plus court que le chemin négatif de mêmes extrémités.

Toutefois cette technique est souvent mise en défaut.

La distance d'inférence

Tourezky a étendu le travaux de Fahlman en introduisant notamment une relation d'ordre sur les chemins d'héritabilité, la *distance d'inférence*, elle est devenu finalement la *relation de préemption*:

> Un chemin d'héritabilité négatif $a \leftarrow \cdots \leftarrow b \not\leftarrow o$ **prime** sur un chemin d'héritabilité positif $a \leftarrow \cdots \leftarrow c \leftarrow o$ si et seulement s'il existe un chemin $a \leftarrow \cdots \leftarrow b \cdots \leftarrow c$ où b est une spécialisation de c.

[8] Un arc de transitivité relie deux objets non consécutifs d'un chemin d'héritage

Donc une exception porte sur tous les ascendants d'un objet excepté, sauf ceux qui sont hérités par ailleurs.

Ces deux techniques semblent faciliter les problèmes d'héritage, En général, on s'arrange pour faire hériter des propriétés des sur classes, même si celles-ci sont connectées par plusieurs sur-classes intermédiaires. A l'inverse pour deshériter un objet des propriétés d'une surclasse, on crée un arc exceptionnel qui relie directement ces deux concepts.

Si un chemin d'héritabilité est arbitrairement choisi:

- S'il s'agit d'un chemin positif, l'exception est dite *relative* car elle peut quand même être effective, dans certain cas.
- S'il s'agit d'un chemin négatif, l'exception est dite *absolue* car elle est toujours effective.

4.12 L'intelligibilité

- Le petit nombre des concepts en facilite la mie en oeuvre et la manipulation.
- L'envoi de messages tous de mêmes structures rend les programmes plus compréhensibles.
- L'héritage renforce la modularité et la possibilité d'approche incrémentale de la compréhension d'un système sauf s'il y a des exceptions. On peut spécifier des objets avec de plus en plus de détails. A la racine de l'arbre d'héritage, on a les structures de données et les méthodes les plus générales, aux éléments terminaux, les structures les plus spécialisées.
- On dispose d'un bon environnement de programmation tel que LISP et d'un langage extensible.
- Un objet est compact, ce qui facilite l'identification des erreurs.
- la réutilisation des logiciels permet beaucoup d'économie.

4.13 Les défauts

- En général, les langages offrant de grandes facilités d'utilisation sont en contrepartie assez peu efficaces en temps d'exécution. Les langages orientés objets ne font pas exception à cette règle. Leur souplesse et leurs possibilités reposent sur une organisation interne relativement complexe, formée de liens entre les descriptifs des objets. Le maintien de ces structures et leur exploitation sont coûteux, particulièrement lors du passage de message où il faut passer de l'instance à la classe puis rechercher la méthode correspondante. Le passage de message est le seul moyen de communication entre objets; il est donc fréquent. Il s'ensuite qu'un langage orienté objets n'est pas très rapide.
- Ces langages sont consommateurs d'espace mémoire, par leur faculté de créer dynamiquement des structures représentant des classes et des instances.

Si la représentation est loin d'être optimale, aussi bien pour l'espace mémoire occupé que pour le temps d'accès aux informations, elle est en contrepartie des plus faciles à exploiter.

De nombreux langages sont aujourd'hui opérationels et commercialisés. Les techniques d'IA y font largement appel: représentation de la connaissance, systèmes experts, systèmes de fenêtrage ... Ils s'intégrent dans des progiciels tels qu'outils de développement de systèmes experts, simulateurs, gestions de base de données, dans des langages plus classiques (C, C++, Lisp, Prolog, Ada). Le système d'exploitation des machines LISP est également construit sur un système d'objets: Les FLAVORS, et le Macintosh d'Apple en est la première application grand publique.

Le style `orienté objet` est une lame de fond qui déborde des applications usuelles d'IA. Elle va modifier, à terme, les habitudes de programmation de tous les secteurs informatiques.

4.14 Comparaison des formalismes

Quels formalismes utiliser lors de la conception d'un système expert? Quels en sont les avantages et inconvénients? Quels critères doit-on considérer pour les discriminer? Il semblerait que:

- puissance d'expression et efficacité;
- aisance à modéliser les concepts que l'on désirer exprimer;
- possibilités offertes pour assister le concepteur de la base dans son travail;

soient les critères les plus déterminants dans le choix d'un formalisme de représentation des connaissances.

Dans les règles de production, la connaissance d'un concept est morcelée dans la base de connaissances, elle est dispersée sur plusieurs règles différentes qui décrivent les attributs et les propriétés particulières du concept.

Notre proposition est que l'extraction des connaissances soit orientée en fonction de la méthode de résolution qui les exploitera.

Il existe plusieurs types de programmation: fonctionnelle, procédurale, par règles, avec objets; il semble qu'aucun de ces types ne constitue à lui seul une réponse au problème, la solution réside certainement dans l'utilisation conjointe de ces techniques.

En intelligence artificielle, la représentation des connaissances conduit à combiner un ensemble de structures de données munies de leur mode d'emploi: des procédures chargés de les interpréter. Les recherches dans ce domaine tendent à développer plusieurs classes de structures de données. C'est ensuite l'utilisation de ces structures qui permet de réaliser un raisonnement, et donc d'entrevoir un processus "intelligent".

On associe donc plusieurs concepts de bases:

- La programmation orientée objets,
- La programmation par règles de production,
- La programmation dans un environnement fonctionnel (langage Lisp).

En général, l'interprétation des connaissances dans un système informatique comme système expert requiert principalement trois phases: définition des modèles de représentation, acquisition de nouvelles connaissances, et enfin raisonnement sur les modèles définis. Le modèle de représentation des connaissances d'un moteur d'inférences dépend étroitement de la classe des problèmes qu'il est amené à traiter, et de son domaine d'application privilégié (un moteur n'est jamais réellement universel). La puissance d'expression du langage (facilité pour l'expert de transmettre son savoir, et complexité des connaissance mises en jeu) est directement liée à la qualité du mode de représentation des connaissances.

Nous avons accepté l'approche fondée sur la représentation sous forme d'objets. Nous mettons à profit l'envoi de messages pour écrire des programmes modulaires et extensibles.

Il est naturel de représenter les règles par des objets, cette approche permet alors la synthèse des méthodes de programmation par objet et par règles.

On profite des systèmes procéduraux dans un langage orienté objet pour effectuer, grâce à l'association de règles aux modèles d'objets, des expertises ponctuelles, déclencher des procédures algorithmiques là où la représentation de la connaissance n'est pas satisfaisante.

On utilise alors le système expert, qui permet une programmation plus indépendante de la représentation des données pour effectuer des expertises globales.

De part ces différences et leur complémentarité, on peut estimer qu'un bon prototype de système pour gérer les opérations de la G.C. devrait s'appuyer d'une part sur un langage orienté objet pour les jugements ponctuels et les déclenchements d'algorithme, et d'autre part sur un moteur d'inférences chargé d'effectuer les expertises globales, travaillant sur la même base de données interprétées.

Chapitre 5

Lisp et objets

5.1 Intérêt de programmer en Lisp

Il existe sur le marché de nombreux langages orientés objets. Au point de vue d'IA, on a préféré la famille **Lisp**: les langages à objets à partir d'un interprète **Lisp**. **Lisp** est l'un des principaux outils de programmation en I.A.. Conçu par J. McCarthy (au M.I.T) en 1958, **Lisp** est aussi un des plus vieux langages de programmation en usage aujourd'hui. Le champ d'application privilégié de **Lisp** reste la programmation symbolique, mais son utilisation hors de l'IA est de plus en plus large.

Les qualités:

- un environnement de programmation agréable,
- la richesse en moyen de représentation et de manipulation symbolique,
- la simplicité et homogénéité de la syntaxe: elle repose sur la seule structure de liste, peu de mots-clés, uniquement sur un ensemble de fonctions,
- le récursivité facilite la programmation des algorithmes complexes.
- la puissance du langage, et en particulier l'équivalence complète entre données et programmes,
- la souplesse: gestion dynamique des objets et des ressources,
- l'interactivité: **Lisp** est principalement interprété, certaines versions complètes sont très efficaces.
- la facilité d'extension, **Lisp** est un langage méta-circulaire, qui peut se définir lui-même, donc être étendu par de nouvelles constructions,
- extensions syntaxiques par macros.

Les défauts:

- le programme **Lisp** manque de concision,
- syntaxe trop élémentaire,

- occupe beaucoup de mémoire,
- langage non typé.

La contribution de **Lisp** à la programmation objet est considérable avec l'influence de **Smalltalk**, il a servi de base de développement à maint langages à objet, qui réutilise directement l'interprète et l'environnement du système **Lisp** choisi comme hôte. Les dialectes modernes de **Lisp**, comme **Le-Lisp** et **Common Lisp**, ont peu à peu intégré une composante objet.

5.2 Ceyx

On a pris **Ceyx**, comme notre langage de travail, non pas pour un choix rigoureux, mais par sa préexistence. Il ne peut peut-être pas concurrencer des produits comme **Smalltalk** ou **C++**, développés et maintenus industriellement. Il s'agit plutôt d'un ensemble d'outils permettant de perfectionner les fonctionnalités de l'interprète **Le-Lisp**, sans perturber son fonctionnement habituel ni son efficacité.

Ceyx, une composante objet de **Le-Lisp**, est un langage orienté objet du type *classe* décrit au paragraphe 4.3, n'ayant que l'héritage simple. Un sous-ensemble permet de programmer dans un style très proche de celui de **Smalltalk**. Il est basé sur la notion de *package*. Les *packages* forment une hiérarchie arborescente dont la racine est le *package* global **()** par défaut. Un *package* est lui-même désigné par un symbole qui peut à son tour appartenir à un autre *package*, et ainsi de suite. un *package* est précédé par le caractère **#:** tel que **#:voie**, les noms des *packages* étant séparés par le caractère **:**, par exemple, **voie** ou **#:voie** désigne le symbole **voie** du *package* **()**, **#:voie:routiere** désignent le symbole **routiere** du *package* **#:voie** et **#:voie:routiere:noeud** désigne le symbole **noeud** du *package* **#:voie:routiere**, alors que **#:noeud** désigne le symbole **noeud** du *package* **()**. (Figure 5.1).

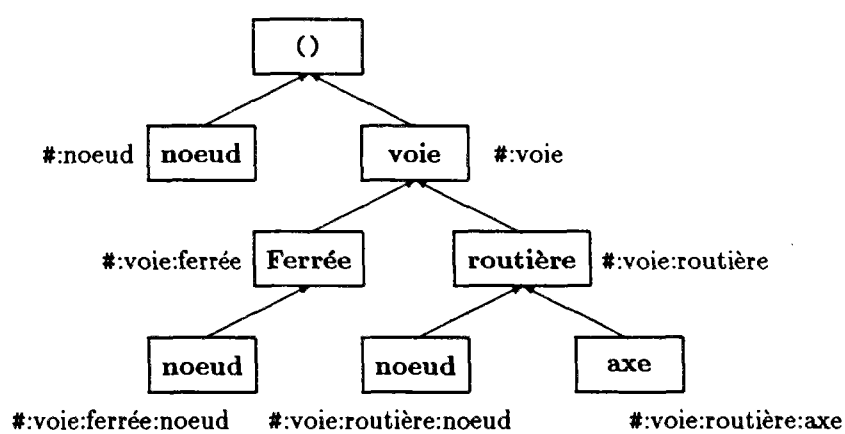


Figure 5.1: La hiérarchie des *packages*

Cette organisation est dans une certaine mesure comparable à celle des fichiers dans un système d'exploitation comme **Unix**, les *packages* constituant l'équivalent des répertoires. Elle permet classiquement d'éviter les conflits de noms entre les variables

appartenant à des modules écrits séparément; comme un nom de fichier, la chaîne d'accès à un symbole décrit le chemin menant au symbole dans l'arbre des *packages*.

Le nom d'un *package* étant aussi un symbole, une valeur peut lui être affectée.

```
? (setq #:voie '(A1 A2 F1))
= (A1 A2 F1)
? #:voie
= (A1 A2 F1)
? (setq #:voie:routiere '(A1 A2))
= (A1 A2)
? #:voie:routiere
= (A1 A2)
? (setq #:voie:ferree '(A1 A2))
= (F1)
? #:voie:ferree
= (F1)
```

Les classes et instances

Un *modèle* est une structure de données définie comme produit étiqueté de structures de données primitives **Le-Lisp** telles ques **atom**, **symbole**, **string**, **cons**, **tcons** (cellules de listes étiquetées) et **vecteurs**. Il est composé de champs, à la manière d'un *record* en **Pascal** ou d'un *struct* en **C**.

La représentation du modèle est stockée dans le champ spécial **O-VAL** du symbole désignant le modèle.

Le symbole désignant le modèle représente également le *package* dans lequel sont définies des fonctions propres au modèle. Celles-ci constitue l'interface, le mode d'emploi en quelque sorte, du modèle: comment accéder au champs des instances, comment les imprimer, etc. Ce *package* est appelé l'*espace sémantique* du modèle, et les fonctions du *package* sont appelées les *propriétés sémantiques* du modèle.

Un **type** est un modèle spécial dont les instances sont représentée par des cellules de listes étiquetées: **tcons**.

La programmation objet est possible grâce à ces modèles:

- A chaque type est associé un espace sémantique dont les propriétés comprennent, entre autres, des fonctions de manipulation spécifiques aux instances du type.
- Les types sont organisés de manière hiérarchique, un sous-type héritant des propriétés sémantiques de ses parents.
- Tous les objets sont autotypés: le type d'un objet, c'est à dire la chaîne d'accès à l'espace sémantique de l'objet, constitue le **car** du **tcons** représentant l'objet.
- Les valeurs des champs d'une instance sont stockées dans le **cdr** du **tcons** représentant l'espace.
- Une opération est appliquée à une instance par un envoi de message, réalisé grâce à la fonction **send**. Comme une instance est auto-typée, elle fournit directement le nom de l'espace sémantique où se trouve la fonction requise.

Plusieurs fonctions de définition de types offrent le choix de l'implantation physique des champs. Une *classe* est définie par **deftclass**, les champs correspondent aux variables d'instance et les propriétés sémantiques aux méthodes.

- L'héritage est simple: les classes forment un sous-arbre de racine **Tclass**, elle-même sous-classe de la classe **()**, racine de l'arbre d'héritage.
- Une classe est représentée par le symbole de même nom que la classe, ce symbole appartenant au *package* de même nom que la superclasse directe: la hiérarchie des classes coïncide avec la hiérarchie de leurs *packages*.
- Les méthodes sont des fonctions définies dans le *package* de leur classe d'appartenance. Les fonctions prédéfinies **Le-Lisp**, telles que **car** ou **cons**, appartiennent à la classe **()**. Elles peuvent être masquées dans une classe définie par l'utilisateur.

Par exemple: la définition d'une classe **Moteur-d-inference** possédant quatre variables d'instance, **base-de-faits**, **base-de-regles**, **base-de-faits-initiale**, **base-de-regles-valides**:

```
? (deftclass Moteur-d-inference
                base-de-faits-initiale
                base-de-faits
                base-de-regle
                base-de-regle-valides)
= #:Tclass:Moteur-d-inference
```

Chaque variable peut posséder un type et une valeur initiale optionnels. Par défaut, la valeur initiale est **()**. L'accès à une variable de classe et réalisé par envoi de message, une écriture se distinguant d'une lecture par un argument supplémentaire donnant la nouvelle valeur de la variable. Toutefois, le receveur du message peut indifféremment être une instance de la classe ou la classe elle-même.

Cette classe est dans le *package* **Tclass**: **#:Tclass:Moteur-d'inference**, le nom de la classe devient l'abbréviation de la chaîne d'accès à ce *package*. Le prototype a la forme suivante:

```
#(#:Tclass:Moteur-d-inference . #[() () () ()])
```

qui est une cellule **tcons** dont le **car** est son type **#:Tclass:Moteur-d-inference**, le **cdr** est un vecteur contenant les valeurs initiales des variables d'instance, dans l'ordre où celles-ci sont données dans la définition de la classe.

On peut ensuite définir des méthodes telles que:

```
Initialiser
Mode-d'inference
Chainage
Afficher-regles-valides
Inferer
```

etc. On peut envoyer des messages en respectant la syntaxe suivante:

(**send** sélecteur *argument1 argument2* ...)

Par exemple, on lance le moteur en chaînage avant en lui fournissant la base de faits initiale **'(f1 f2 f3)**, et la base de règles **'(r1 r2 r3)** comme arguments:

```
? (sendq en-avant mi '(f1 f2 f3) '(r1 r2 r3))
= (f1 f2 f3 f4) ; la base de faits finale
```

L'héritage des variables est statique: à la création d'une classe, le vecteur du prototype d'instance de la superclasse est dupliqué et augmenté des valeurs des variables supplémentaires introduites par la classe. Le rang des variables héritées ne change pas, et les méthodes d'accès définies dans les superclasses restent valables pour toutes les instances des sous-classes.

En revanche, l'héritage des méthodes est dynamique. Lors d'un envoi de message, la classe de l'objet receveur est déterminée grâce à la fonction prédéfinie **type**. Celle-ci retourne en fait le type de son argument, avec la convention que le type d'une cellule **tcons**, donc d'une instance d'une classe est donné par son **car**.

```
? (type mi1)
= #:Tclass:Moteur-d-inference
```

La recherche de la méthode à appliquer s'effectue en remontant l'arbre d'héritage à partir de la classe obtenue précédemment. Elle s'arrête dès qu'une fonction de même nom que le sélecteur est trouvée dans le package associé à la classe courante.

```
? (sendq cons 'le 'lisp)
  ; la fonction 'cons' trouvee dans le package ()
= (le . lisp)
```

Ceyx donne aussi la possibilité de définir des méthodes dont la sélection fait intervenir les deux premiers arguments des messages grâce aux **defrule** en précisant la classe d'appartenance des ces deux premiers arguments.

Variable de classe

Comme l'implantation de Ceyx est simple, homogène et ouverte, il est facile d'ajouter ses propres utilitaires ou de perfectionner ceux qui existent, sans nuire à l'intégrité du langage.

A titre d'exemple, nous proposons d'implanter des *variables de classe*, au sens Smalltalk, partagées par toutes les *instances* d'une *classe*, et de ses sous-classes, au lieu de donner une valeur à une *variable d'instance* à chaque fois qu'on crée une nouvelle instance.

Par exemple, elles pourraient servir à représenter la propriété : le **signe conventionnel** pour toute la classe **autoroute**, commun à toutes les instances de cette classe.

Au moyen d'une procédure-variable

Une première méthode est d'emprunter une procédure d'une classe. Une visite de la valeur d'une variable d'instance (champ) et une activation d'une procédure se font toutes deux par le même moyen: envoi de message avec un sélecteur plus éventuellement des arguments s'il s'agit d'une activation d'une procédure. On peut donc très bien utiliser une procédure dont la fonction est de juste stocker une valeur; cette procédure ressemblera ainsi à une simple variable, d'où le nom *procédure-variable*.

Une procédure-variable est partagée par toutes les instances de la classe et celles de ses sous-classes et sera héritée dynamiquement. Elle peut être redéfinie au niveau des sous-classes sans déranger les sur-classes.

On définit une classe **Autoroute** avec trois variables d'instance: **nom, largeur, voisins**.

```
? (deftclass  Autoroute  nom largeur voisins)
= #:Tclass:Autoroute
? (defmake {Autoroute}
              make-auto (nom largeur voisins))
= make-auto
```

On peut donc créer une instance d'autoroute: **a1** avec la fonction de création d'instance: **make-auto**.

```
? (setq a1 (make-auto 'a1 20 '(a2 a3)))
= #(#:Tclass:Autoroute . #[a1 20 (a2 a3)])
```

On souhaite donner le signe conventionnel **(20 30 20)** pour représenter toutes les autoroutes à l'échelle de 1:1 000 000. On définit donc une procédure-variable:

```
? (de {Autoroute}:signe-cv (autoroute)
     '(20 30 20))
= #:Tclass:Autoroute:signe-cv
```

Par un envoi d'un message, on peut obtenir le signe conventionnel de l'autoroute **a1** à l'échelle de 1 : 1 000 000.

```
? (sendq signe-cv a1)
= (20 30 20)
```

Si une autoroute européenne (supposons son existence) **a11** est une instance de la classe **Auto-euro** qui est une sous-classe de la classe **Autoroute**:

```
?  (deftclass {Autoroute}:Auto-euro )
= #:Tclass:Autoroute:Auto-euro
? (defmake {Auto-euro} make-euro (nom largeur voisins))
= make-euro
```

```
? (setq a11 (make-euro 'a11 20 '(a1)))
= #(#:Tclass:Autoroute:Auto-euro . #[a11 20 (a1)])
```

Normalement, par l'héritage de propriété de la sur-classe **Autoroute**, **a11** a le même signe conventionnel qu'une autoroute normale telle que **a1**:

```
? (sendq signe-cv a11)
= (20 30 20)
```

Pour qu'une autoroute européenne soit représentée autrement qu'une autoroute normale, on peut redéfinir le signe conventionnel de la classe **Autoroute-euro** par (20 10 10 10 20):

```
? (de {Auto1}:signe-cv (auto-euro)
?     '(20 10 10 10 20))
= #:Tclass:Autoroute:Auto-euro:signe-cv
? (sendq signe-cv a1)
= (20 30 20)
? (sendq signe-cv a11)
= (20 10 10 10 20)
```

Le même signe conventionnel représentant **a1** n'a pas été modifié.

Au moyen d'une vraie variable de classe

La définition des variables de classe permet une programmation plus déclarative que procédurale; dans ce dernière, on est parfois obligé de vérifier le type de la classe d'un objet avant de lui donner une valeur cherchée.

On peut définir une variable de classe en deux étapes séparément: on reprend la même classe de **Autoroute** défini précédemment.

- Initialisation de la variable avec la valeur donnée:

```
? (setq {Autoroute}:couleur
        'rouge)
= rouge
```

Comme pour une variable d'instance toujours, une variable de classe telle que **couleur** de la classe **Autoroute** et la fonction d'accès correspondante: **{Autoroute}:couleur** portent le même nom; elles sont implantées toutes deux dans le *package* de leur classe d'appartenance: **#:Tclass:Autoroute** par exemple. Une variable de classe ne fait cependant pas partie du prototype d'instance détenu par la classe et n'est donc pas dupliquée lor des créations d'instances:

```
? (setq a1 (make-auto 'a1 2 ()))
= #(#:Tclass:Autoroute . #[a1 2 ()])
? (send 'couleur a1)
** send : Undefined semantic :
        (#:Tclass:Autoroute couleur)
```

- Définition de la méthode d'accès à la variable sur le modèle des méthodes d'accès aux variables d'instance:

```
? (de {Autoroute}:couleur (self . valeur)
   (if valeur  ; une ecriture
       (setq {Autoroute}:couleur (car valeur))
             ; une lecture
       {Autoroute}:couleur))
 = #:Tclass:Autoroute:couleur
```

Afin de pouvoir simuler un envoi de message à la classe, le symbole représentant la classe est ensuite valué avec une instance de la classe:

```
? (setq {Autoroute} (omakeq Autoroute))
= #(#:Tclass:Autoroute . #[() () ()])
? {Autoroute}
= #(#:Tclass:Autoroute . #[() () ()])
? (type {Autoroute})
= #:Tclass:Autoroute
```

Cette fonction ramène en valeur une instance du modèle **{Autoroute}** qui est une instance de défaut de la classe **{Autoroute}**. Le rôle de cette instance étant de fournir à la fonction **send** le *package* à partir duquel est recherchée la méthode d'accès à la variable de classe; seul son type, donné par son **car**, est utile. Le **cdr**, c'est à dire le vecteur regroupant les valeurs des variables d'instance, peut donc être supprimé et remplacé par la liste vide:

```
? (setq {Autoroute}
        (tcons '{Autoroute} ()))
= #(#:Tclass:Autoroute)
? {Autoroute}
= #(#:Tclass:Autoroute)
? (type {Autoroute})
= #:Tclass:Autoroute
```

[15] propose une définition de la fonction devarclass permettant de réaliser automatiquement les différentes opérations à partir des couples (**variable . valeur**)s qui lui sont donnés en paramètres:

```
(dmd devarclass (class . var-vals)
 '(progn
  ,@(mapcan ; pour tous les couples var-vals
```

```
      (lambda (var-val)
       (let ((sym (symbol class (car var-val))))
            ; la variable dans le package de la classe
        (list
           ; initialisation de la variable de classe
         '(setq ,sym ,(cadr var-val))
           ; cr\'eation la fonction  d'acc\'es
         '(setfn ',sym expr '((self . valeur)
            (if valeur (setq ,sym (car valeur)) ,sym))))))
     var-vals)
    ; affectation de la fausse instance a la classe
  (setq ,class ',(tcons class ())))
```

Ensuite on peut créer des variables **couleur**, **type** **signe-conv** de classe pour toutes la classe **{Autoroute}**:

```
? (devarclass {Autoroute}
?      (sign-conv '(20 30 20))
?      (type      'autoroute)
?      (couleur   'rouge))
= #(#:Tclass:Autoroute)
```

On peut visiter les valeurs d'une instance **a1**:

```
? (sendq couleur {Autoroute})
= rouge
? (sendq couleur a1)
= rouge
```

ou donner une nouvelle valeur:

```
? (sendq couleur {Autoroute} 'jaune)
= jaune
? (sendq couleur a1)
= jaune
```

On peut également donner une nouvelle valeur à une variable par l'intermédiaire d'une vraie instance **a1** au lieu de la fausse instance **{Autoroute}**, mais qui sera également une nouvelle valeur de toutes les autres instances, ce qui justifie le caractéristique de **variable de classe**:

```
? (setq a2 (make-auto 'a2 '23m '(a1)))
= #(#:Tclass:Autoroute . #[a2 23m (a1)])
? (sendq couleur a2)
= jaune
? (sendq couleur a2 'verte)
= verte
? (sendq couleur a1)
= verte
? (sendq couleur {Autoroute})
= verte
```

On peut également redéfinir la valeur d'une variable de classe dans les sous-classes sans changer celle de la sur-classe, soit **Autoroute-prive** est une sous-classe de la classe **Autoroute**, et ap_1 en est une instance

```
? (deftclass {Autoroute}:Autoroute-prive )
= #:Tclass:Autoroute:Autoroute-prive
? (type ap1)
= #:Tclass:Autoroute:Autoroute-prive
? (sendq couleur ap1 'vert)
=  vert
? (sendq couleur {Autoroute})
= rouge      ; comme precedent, sans changer
```

Une classe qui n'a qu'une seule instance n'a aucun intérêt particulier d'avoir des variables de classe. Un changement de structure (par exemple, un ajout des variables d'instance) de la classe n'entraîne qu'un minimum de changement de l'instance déjà créée avant le changement de la structure de sa classe mère. Mais l'ajout des variables de classe à une classe, même avec de multiples instances créées ne nécessite aucun changement pour les instances créées auparavant. Dans notre implantation, une variable de classe d'une classe, comme une variable d'instance, reste une variable de classe dans les sous-classes aussi loin que l'héritage de classe à sous-classe puisse fonctionner.

Pour Ceyx, il s'agit plutôt un ensemble d'outils permettant de perfectionner l'interprète **Le-Lisp**, sans perturber son fonctionnement habituel ni son efficacité. Toute la puissance des constructions **Lisp** est conservée pour l'écriture des méthodes, qui utilise indifféremment les appels fonctionnels classiques et envois de message.

5.3 Common Lisp Object System: CLOS

Common Lisp comporte déjà quelques constructions objets, avec notamment la fonction **defstruct**, comme en **Le-Lisp**, pour définir des structures analogues aux enregistrement **record** Pascal. L'option **:include** permet de réaliser un mécanisme rudimentaire d'héritage simple de variables, mais la notion de méthode n'existe pas.

CLOS est un véritable système objet, avec une syntaxe héritée de Common Lisp, un mécanisme d'héritage multiple fondé sur un algorithme de linéarisation, des fonctions génériques unifiant l'appel de fonction et l'envoi de message, ainsi que des mécanismes de combinaisons de méthodes et de redéfinition dynamiques des classes inspirés de **New Flavors**.

Classe et instanciation

Une classe regroupe un ensemble de *slots* qui sont des variables d'instance possédant des propriétés. Elle se définit avec la fonction **defclass**:

```
(defclass Autoroute
          (Routea2chaussees)     ; les superclasses
  ((nom                          ; le premier slot
     :allocation :instance       ; variable d'instance
```

```
    :type string)                ; le type du slot

  (couleur
    :allocation :classe          ; variable de classe
    :type string
    :initform 'noir)             ; valeur initiale
  (largeur
    :type fix
    :initarg :largeur) ;autorise l'initialisation when
  (axe                 ; creation d'instance
    :type cons
    :initarg :axe
    :reader lire-axe)  ;font. de lecture de la valeur de axe
  (larcs
    :type cons
    :initarg :larcs)
  (lvoisins
    :type cons
    :initarg :lvoisins)
  (lncs
    :type cons
    :initarg :lncs))
  (:documentation ''ING 1990''))      ;commentaire
```

Pour créer une instance d'une autoroute avec **A4** comme nom, largeur non encore connue (de valeur: ()), (p1 p2) comme axe, avec a1 comme arc, voisins non encore connu, **s1** comme noeud:

```
(setq  A4
   (make-instance
        'Autoroute 'A4 :largeur () '(p1 p2) '(a1) '(s1)))
```

La redéfinition d'une classe est possible, le système se charge de modifier en conséquence les instances de la classe, ainsi que ses sous-classes et leurs instances.

Les fonctions génériques

En général la sélection de la méthode invoquée par un envoi de message est fonction de la seule classe du receveur qui est le premier argument du message. C'est sur ce premier argument qu'est fait le décodage de type permettant de trouver la fonction à appliquer en profitant des informations sur le reste des arguments. CLOS permet d'avoir plusieurs receveurs classes, c'est-à-dire que la sélection se fait pour chaque argument du message: l'objet receveur n'est plus différencié des autres arguments du message.

Par exemple, la même opération **écarter** peut être appliquée:
à trois carrefours:

```
(ecarter carf1 carf2 carf3)
```

à trois routes:

```
(ecarter route1 route2 route3)
```

ou encore à une route et deux carrefours:

```
(ecarter route1 carf2 carf3)
```

Le résultat sera différent en fonction de produit cartésien des classes de ses arguments. Il est possible d'obtenir ce résultat en Lisp pur en testant le type des arguments, mais CLOS réalise ce travail automatiquement en utilisant la notion de fonction générique. Un problème d'efficacité se pose quand le nombre des arguments s'accroît pour la définition des méthodes d'une fonction générique et pour assurer le déclenchement effectif de la méthode désirée. Cette notion de fonction générique se trouve également en Ceyx décrit ci-avant, avec les bi-méthodes, la sélection de méthode se fait en fonction du type des deux premiers arguments du message.

5.4 KEE

Il existe de nombreux outils informatiques pour aider les concepteurs de logiciels, en particulier de systèmes experts: des générateurs des systèmes experts tels que KEE, ART, SMECI. Il était question qu'on se serve d'un de ces outils pour le travail de cette thèse. Malheureusement, l'auteur n'a eu l'occasion de se familiariser de KEE qu'à la fin de cette thèse.

Cependant KEE mérite d'être mentionné pour son ensemble d'outils conçus pour aider les concepteurs de systèmes experts.

Développé autour de UNITS, Le système KEE est un système hybride dans le sens où il associe plusieurs concepts de bases de l'IA: la programmation orientée objets (développé autour de UNITS), la programmation par règles de production (la programmation logique avec les principes d'unification et d'association a été associée), dans un environnment fonctionnel avec le langage Lisp. La version 3 sur station de travail SUN est basée sur Sunview, disposant d'une riche interface orientée fenêtre et graphique.

L'objectif d'un tel outil est de fournir au concepteur les outils informatiques lui permettant de se focaliser plus sur ses problèmes que sur la maîtrise des langages de programmation. Pour cela il permet, en principe:

- de modéliser rapidement un prototype de système expert,
- d'être facilement manipulable pour qu'un expert puisse sans un long apprentissage, construire l'ensemble de ses règles et de ses objets,
- de faire évoluer le prototype de système final robuste,
- d'être ouvert sur l'environnement Lisp de la machine hôte, pour autoriser de finitions à la convenance de l'utilisateur.

En tant que langage objets, la particularité de KEE réside dans la définition de l'instanciation et de l'héritage. Les deux notions sont, en effet, regroupées en un même mécanisme de transmission d'information entre objets. Ce mode de transmission est très sophistiqué et peut être paramétré à l'aide d'un grand nombre d'opérateurs.

Dans la pratique, KEE est assez lourd, sa maîtrise complète demande la connaissance du langage Lisp et un investissement important. Il est beaucoup utilisé pour construire des maquettes et peu de systèmes opérationnels sont connus.

Partie III

Réalisation

Chapitre 6

Modélisation des opérations graphiques

L'histoire nous montre que l'évolution de la cartographie a toujours tendu à une nécessaire adaptation aux besoins des usagers toujours plus nombreux et plus exigeants en utilisant les techniques de pointe les plus efficaces en matière de conception, de rédaction, de reproduction et d'impression.

L'évolution actuelle passe par une automatisation du recueil de l'information, du traitement des données et de leur transcription graphique.

Le premier obstacle est la modélisation et quantification de tous les tâches du processus de la généralisation. Ceci constitue purement un problème cartographique plutôt qu'un problème informatique. Ce chapitre va dans cette direction en discutant les divers problèmes pour identifier et classer les problèmes et en proposant des solutions.

Nous avons noté que la représentation graphique, moyen d'enregistrement, de traitement et de communication de l'information géographique, est un langage qui a son vocabulaire et sa grammaire, donc, par conséquent, ses règles lexicales et syntaxiques.

Que, jusqu'à présent, ces règles aient été appliquées inconsciemment et sans formulation n'implique nullement leur inexistence et l'impossibilité de les énoncer sous une forme mathématique.

Nous avons constaté au chapitre 2 que, en réalité, les propriétés des variables visuelles s'expriment sous forme de longueur, d'équivalence, d'ordre ou de rapports numériques, et que les règles de lisibilité s'énoncent en termes numériques.

Peu de modélisation structurale a été faite dans le cadre de la généralisation automatique. *La difficulté de l'automatisation de la rédaction ou de la généralisation cartographique ne réside pas dans le découpage en fonctions ni même dans l'écriture de ces fonctions, mais surtout dans leur mise en oeuvre* [8, page 23]. En fait, les principes de la généralisation cartographique sont bien beaux à raconter, mais difficiles à réaliser.

Nous essayons dans ce chapitre de montrer quelques aspects de modélisation des principes des opérations de la généralisation, qui sont justiciables d'une formulation plus ou moins mathématique si on prend la peine de procéder de façon logique et

méthodique. Certaines opérations sont globales, d'autres sont locales.

Dans les chapitres suivants, on abordera quelques réalisations sur des exemples réels.

Vue les différences et les qualités que la méthode procédurale et la méthode déclarative en informatique peuvent rapporter, on essaye de modéliser des processus procéduraux et des formalismes déclaratifs pour pouvoir bénéficier des avantages de deux méthodes.

6.1 Les mesures

Les notions ci-dessous, toutes relatives à la notion générale de mesure, sont à la base de la base de reconnaissance des formes des objets géométriques et de leur relations. La liste ci-dessous n'est pas exhaustive. Nous reviendrons sur ces notions dans la suite de l'exposé.

Mesure de densité : cette mesure est évaluée pour plusieurs objets. Il s'agit plus précisément de nombre d'objets ponctuels, linéaires ou zonaux par unité de surface.

Mesure de distribution : cette mesure concerne la distribution de l'ensemble des objets de la carte. On peut estimer la dispersion des objets ponctuels d'une même catégorie par rapport à leur barycentre (voir la section 6.5.3).

Mesure de longueur : cette mesure concerne les objets linéaires tels que les voies de communication.

Mesure de sinuosité : comme la mesure précédente, elle concerne les objets linéaires et renseigne sur les changements d'angle entre les segments de cet objet.

Mesure de forme : la forme joue un rôle important pour déterminer si un objet est représentable à une nouvelle échelle. Cette mesure concerne les géométries des points, lignes, ou surfaces, etc.

Mesure de Distance : La distance entre les objets élémentaires géométriques tels que: points, lignes, surfaces peut être évaluée par la plus courte distance perpendiculaire selon leur formes. Cette mesure de distance indique si la généralisation est nécessaire si des conflits apparaissent après une réduction d'échelle.

Certaines mesures sont faciles à déterminer et évaluer dans le cadre numérique, d'autre non. Il existe d'autres mesures abstraites concernant la distribution spatiale telles que homogénéité, symétrie, complexité. Ces dernières sont difficiles à calculer. Toutes les mesures sont en fonction de l'échelle, du but de la carte, etc. Elles sont à la base des opérations de la GC.

6.2 Règles de sélection

6.2.1 Cas général

L'aspect sélectif de la généralisation intervient chaque fois qu'il y a une réduction d'échelle, ou changement de but ou de thème de la carte. Les graphismes occupent

toujours une place proportionnellement égale ou plus souvent beaucoup plus grande que dans la réalité. Par conséquent, soit par option, soit par manque de place, en général, tout objet est menacé par la peine de mort:

Règle-de-sélection 1 *Tout objet a le droit de rester tant qu'il y a de la place et risque sinon d'être éliminé.*

On doit s'assurer qu'un tribunal se charge de juger avant de condamner suivant sa propre loi. Cette loi est constituée par les règles ou critères de sélection.

Au niveau de la conception, la sélection se traduit, d'une part par une réduction des catégories et des classes d'objets représentés, qui se traduit, soit par l'abandon des classes inférieures, soit par l'élargissement des intervalles séparant les paliers conservés, en valeurs quantitatives ou qualitatives, soit par une modification de l'implantation au niveau de schématisation.

Par exemple, sur le tableau 2.3 de la page 28, il existe une douzaine de classes de routes et chemins sur la carte de France à l'échelle 1/25 000, et 4 seulement sur la carte à 1/500 000 où les routes principales (classement qualitatif) regroupent dans un signe unique, les routes principales de plus de 7 m et de moins de 7 m (classement quantitatif) différenciées aux échelles plus grandes.

Le problème est de choisir les objets à conserver ou à éliminer. La tendance est en général de disposer au mieux de la surface du papier disponible, c'est à dire de porter autant de détails que possible, ce qui risque d'occuper la place nécessaire aux objets essentiels.

Il est difficile de traiter ce problème sans une vision d'ensemble des détails de même nature considérés indépendamment des autres catégories de détails; on trouve ici une justification des préparations de généralisation qui sont, en réalité, des préparations de sélection; elles correspondent aux problèmes suivants:

- choix des catégories des objets à représenter sur la carte,
- classement selon de nouvelles catégories ou de nouvelles classes des détails hiérarchisés adaptées à l'échelle de la carte,
- choix des détails ponctuels, linéaires et zonaux à représenter parmi ceux qui font l'objet d'une figuration facultative,
- sélection et position des cotes et des écritures, etc.

On a une toute première règle de sélection quantitative :

Règle-de-sélection 2 *Tout objet qui est inférieur à la limite du seuil de lisibilté*[1] *est* **éliminable**.

On précise ici que le mot **éliminable** indique l'action de suppression est simplement possible, et non certaine, car d'autres règles plus puissantes pourraient peut être sauver un objet éliminé par une règle moins puissante, ainsi que le précise la règle de sélection qualitative ci-dessous:

Règle-de-sélection 3 *Tout objet est maintenu si son importance le justifie (même s'il était peut-être jugé éliminable par d'autres règles).*

[1] définit dans la section 2.4

On a deux critères principaux:

- **Critère 1**: Spécifications relatives à la carte: qualitatives et quantitatives. On définit les détails à faire figurer par leur nature ou leur importance. Par exemple, on peut supprimer toutes les routes selon un critère:
  - *qualitatif*:
    - non classées nationales ou départementales,
    - non revêtues.
  - *quantitatif*: dont la largeur est inférieure à 5 m,

  Il est possible que les objets habituellement éliminés puissent être maintenus dans une région de faible densité graphique.

  Il est également possible qu'un élément qui doit être éliminé à cause de sa petite surface puisse être maintenu si son importance le justifie.

- **Critère 2**: entre les objets dont la représentation est facultative, on doit essayer de maintenir le caractère régional et les contrastes d'occupation du sol.

Ces deux critères ne sont pas indépendants, ils doivent être bien combinés ensemble. Par exemple, selon le critère1, on supprime beaucoup d'objets quand ils sont nombreux, et selon le critère2 on maintient les objets quand ils sont rares, au risque d'aboutir à une égalisation de l'image cartographique, dénaturant le caractère régional et réduisant les contrastes qui apparaissent dans la nature et sur les cartes de base.

Il est difficile d'échapper à cette tendance si on ne s'astreint pas à respecter strictement des règles fixant le nombre d'objets qui doivent figurer ou disparaître à chaque changement d'échelle, surtout dans la catégorie d'objets à figuration facultative.

Dans la section suivante, on essaye d'établir et formuler des règles de sélection quantitative.

6.2.2 Idées théoriques de sélection

Si on désigne par N_i le nombre d'objets figurant sur la carte initiale dont l'échelle est $1/E_i$, et N_f le nombre d'objets conservés sur la carte dérivée finale dont l'échelle est $1/E_f$, c'est à dire avec $E_f > E_i$, on doit admettre que les relations qui lient N_i et N_f est la suivante:

$$N_f = N_i \sqrt{(\frac{E_i}{E_f})^n} \tag{6.1}$$

quand n varie de 0 à 4, l'équation 6.1 donne des relations différentes.

- **n = 0**: la carte finale est une simple réduction à la photocopieuse de la carte initiale; tous les détails sont conservés:

$$N_f = N_i \tag{6.2}$$

  Ceci correspond à des cartes de très grandes échelles avec une représentation homométrique et une surface de papier amplement suffisante.

- `n = 1, 2, 3`: des relations intermédiaires entre les 2 extrémités: `n = 0` et `n = 4`,

$$N_f = N_i\sqrt{\frac{E_i}{E_f}} \tag{6.3}$$

$$N_f = N_i\sqrt{(\frac{E_i}{E_f})^2}$$

$$= N_i(\frac{E_i}{E_f}) \tag{6.4}$$

$$N_f = N_i\sqrt{(\frac{E_i}{E_f})^3} \tag{6.5}$$

- `n = 4`: la densité des objets est identique sur les 2 cartes: le nombre d'objet N_f est proportionnel au rapport des surfaces de papier et inversement proportionnel au carré du rapport des dénominateurs des échelles, soit:

$$N_f = N_i(\frac{E_i}{E_f})^2 \tag{6.6}$$

cette dernière règle est une caractéristique des cartes générales, à partir de l'échelle de 1/ 1 000 000, lorsque les graphismes sont à la limite de la lisibilité et que la densité maximale est atteinte.

Ces cinq relations se traduisent par les pourcentages de détails conservés dans le tableau 6.1 sur la page 96 quand la réduction d'échelle $\frac{E_i}{E_f}$ varie de 2 à 20.

On voit que quand les échelles sont voisines, c'est à dire que le facteur de réduction d'échelle est moins de 5, la sélection est relativement plus facile, on doit choisir un objet sur deux ou sur 10 sur la carte initiale, la liberté de choix est relativement limitée, on apprécie mieux le choix fait.

Quand le facteur est plus grand, le choix est plus grand, on laisse plus de liberté au dessinateur, ce qui risque augmente d'interprétation individuelle. De plus, dans ce cas, les graphismes à dessiner sont gros, la généralisation est délicate car les humains auront du mal à apprécier l'espace disponible et on peut avoir tendance à surcharger la carte.

Les différences de conception risquent d'introduire un manque d'homogénéité.

Cependant l'utilisateur de la carte ne juge pas en fonction de la densité des objets représentés et n'est pas à même de comparer la densité des objets qui ont été conservés avec celle des objets supprimés C'est pourquoi la plupart des auteurs cartographes qui se sont penchés sur ce problème s'accordent pour adopter la règle de sélection exprimée par Töpfer et Pillewiser sous la forme suivante:

$$N_f = N_i C_b C_z \sqrt{\frac{E_i}{E_f}}$$

C_b est une constante dite d'*amplification symbolique* qui prend les valeurs suivantes :

$\frac{E_i}{E_f}$ =	Détails conservés	Relation $\frac{N_f}{N_i}$ =				
		1	$\sqrt{(\frac{E_i}{E_f})}$	$\frac{E_i}{E_f}$	$\sqrt{(\frac{E_i}{E_f})^3}$	$(\frac{E_i}{E_f})^2$
2	en %	100%	70.7%	50.0%	35.4%	25.0%
	en rapport	$\frac{1}{1}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{1}{3}$	$\frac{1}{4}$
3	en %	100%	57.7%	33.3%	19.2%	11.1%
	en rapport	$\frac{1}{1}$	$\frac{3}{5}$	$\frac{1}{3}$	$\frac{1}{5}$	$\frac{1}{9}$
4	en %	100%	50.0%	25.0%	12.5 %	6.25%
	en rapport	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
5	en %	100%	44.7 %	20.0%	8.9%	4.0%
	en rapport	$\frac{1}{1}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{1}{12}$	$\frac{1}{25}$
10	en %	100%	31.6 %	10.0 %	3.1%	1.0%
	en rapport	$\frac{1}{1}$	$\frac{1}{3}$	$\frac{1}{10}$	$\frac{1}{30}$	$\frac{1}{100}$
20	en %	100%	22.3 %	5.0 %	1.1%	0.25 %
	en rapport	$\frac{1}{1}$	$\frac{1}{5}$	$\frac{1}{20}$	$\frac{1}{100}$	$\frac{1}{400}$

Tableau 6.1: Tableau de pourcentages de détails conservés en fonction du facteur de réduction.

- quand l'amplification conventionnelle correspond au rapport des échelles:

$$C_b = 1$$

- quand le détail n'est pas amplifié:

$$C_b = \sqrt{\frac{E_f}{E_i}}$$

ce qui ramène à la relation 6.2: $N_f = N_i$,

- quand le signe conventionnel correspond à une amplification importante:

$$C_b = \sqrt{\frac{E_i}{E_f}}$$

ce qui ramène à la relation 6.4: $N_f = N_i(\frac{E_i}{E_f})$

C_z est une constante de *forme symbolique*:

- $C_z = 1$, quand les signes conventionnels conservent des dimensions identiques au cours de la généralisation.

- sinon, pour tenir compte des variations de dimensions du signe, épaisseur des lignes ou surface du symbole, entre la carte initiale (D_i) et la carte dérivée (D_f):

$$C_z = (\frac{D_i}{D_f})\sqrt{\frac{E_i}{E_f}}$$

Ce coefficient tient donc compte de l'affinement comme du grossissement éventuel des graphismes utilisés.

Bien entendu, la relation ci-dessus précise le nombre d'objets à maintenir sur la carte dérivée, en laissant au préparateur l'initiative de conserver tel objet plutôt que tel autre, mais elle assure l'homogénéité du choix quantitatif pour toutes les feuilles d'une même série et elle réduit l'influence de l'interprétation individuelle.

Les relations de sélection citées précédemment pour des objets ponctuels se révèlent parfaitement efficaces puisqu'elles conservent les vrais rapports de densité; elles sont peu adaptées dans le traitement des carrefours, puisque ces derniers occupent une surface non négligeable. Mais dans le cas linéaire où les éléments ne sont pas aussi réguliers, elles ne sont qu'une référence plus théorique que pratique.

6.2.3 Cas linéaire

Quand l'épaisseur des signes conventionnels linéaires simples est pratiquement constante, ce qui facilite la codification, le choix se fait habituellement selon des critères de largeur, de longueur, en conformité avec les règles de lisibilité définies dans la section 2.4.

Bien que quantitatif, le critère de longueur n'apporte pas une solution satisfaisante au problème de la généralisation car, sauf cas particulier, il n'est pas en corrélation avec la densité du réseau des voies de communication ou du réseau hydrographique; son application systématique et successive pouvant dénaturer totalement l'aspect géographique local, il vaut mieux éviter de trop se servir de cette règle.

En effet, les éléments linéaires, conservés en vertu de leur nature, peuvent être choisis localement en fonction de leur longueur, de leur largeur, ou de tout autre facteur géographique.

Si $\mathtt{Largeur}_{(route)}$ est la largeur réelle d'une route, on a généralement la règle de sélection purement quantitative (en ignorant son importance) suivante:

Règle-de-sélection 4 *Si*

$$\frac{\mathtt{Largeur}_{(route)}}{E} < 0.1mm \tag{6.7}$$

alors cette route ne sera pas figurée sur la carte à partir de l'échelle $1/E$

Cette règle quantitative est corrigée par la règle qualitative:

Règle-de-sélection 5 *Si l'indice de l'importance d'une route est plus grand que l'indice de seuil de sélection* `I(E)`*, alors elle doit être maintenue sur la carte.*

L'indice de seuil de sélection `I(E)` varie en fonction des spécifications relatives à l'échelle E. Par exemple, sur une carte à partir de l'échelle de 1/1 000 000, seuls les chemin d'exploitation, les sentiers, les ruelles d'agglomération ne sont plus figurés.

Si on définit `Seuildelong{E}` comme le seuil de longueur de route qu'on fait figurer sur une carte à l'échelle $1/E$.

Règle-de-sélection 6 *Soit* $Longueur_{(route)}$ *la longueur réelle de la route, si*

$$\frac{Longueur_{(route)}}{E} < \mathtt{Seuildelong\{E\}} \tag{6.8}$$

cette route ne sera pas figurée sur une carte à l'échelle $1/E$.

	Relation $\frac{N_f}{N_i}$ =				
applicable	1	$\sqrt{(\frac{E_i}{E_f})}$	$\frac{E_i}{E_f}$	$\sqrt{(\frac{E_i}{E_f})^3}$	$(\frac{E_i}{E_f})^2$
aux échelles	$\leq$ 1/5 k	1/25 k - 1/50 k	1/50 k - 1/100 k	1/100 k -1/1M	$\geq$ 1/1 M

Tableau 6.2: Tableau de relation de sélection utilisable pour les échelles

Le `Seuildelong{E}` varie d'une échelle à l'autre, mais on a toujours pour un tronçon ou un segment individuel :

$$\texttt{Seuildelong\{E\}} > 1mm \tag{6.9}$$

Si on connaît uniquement la longueur figurée $Longueur_i$ d'une route à l'échelle $1/E_i$, alors sur une carte à l'échelle $1/E_f$, $E_f > E_i$: on a une règle:

Règle-de-sélection 7 *Si*

$$Longueur\{E_i\} < 0.3\frac{E_f}{E_i}mm \tag{6.10}$$

alors elle éliminable sur la carte à l'échelle $1/E_f$

Les relations de la section 6.2.2 sont valables pour les réseaux linéaires compte tenu de la densité moyenne du réseau, voir le tableau 6.2 dans [1]. On confirme que par expérience, la densité graphique maximale est généralement atteinte pour la carte à 1/1 000 000 et au delà de cette échelle, seule la relation 6.6 est utilisable. Pour les échelles intermédiaires, une solution simple mais satisfaisante, en accord avec la règle de sélection générale, consiste dans l'application de la relation 6.3 entre les échelles de 1/25 000 et de 1/50 000, de la relation 6.4 entre le 1/50 000 et le 1/100 000 et de la relation 6.5 entre 1/100 000 et le 1 000 000.

Le cas zonal est un autre problème, tant aussi complexe! On n'en parlera pas davantage.

6.3 Règles de schématisation structurale

Après la procédure de sélection, on a tous les détails de la future carte. La procédure de schématisation sera une précompilation de la carte en tenant compte de l'ensemble de la carte.

Au stade de la réalisation, après l'aspect sélectif, c'est la schématisation structurale qu'il convient de définir et, si possible, de normaliser.

On se contentera de montrer quelques aspects pratiques sur la schématisation des éléments linéaires; il s'agit spécialement des lignes individuelles comme les axes de voies de communication.

Pour bien apprécier et comparer les opérations, on suppose qu'on travaille sur un fond de la carte initiale de l'échelle $1/E_i$, en prenant les signes conventionnels de la carte finale de l'échelle $1/E_f$, c'est à dire que les signes de la carte finale sont amplifiés $\frac{E_f}{E_i}$ fois.

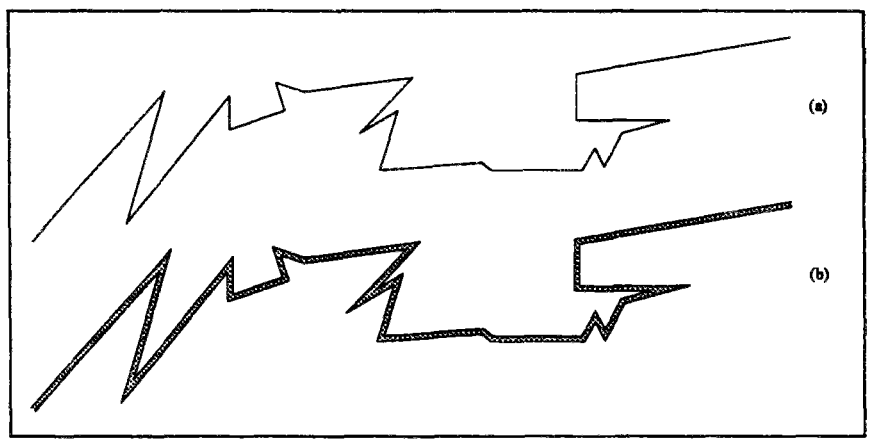


Figure 6.1: Axes de voies de communication

Ces lignes sont matérialisées sur la carte par des traits simples (figure 6.1:a) de largeur finie ou des signes conventionnels linéaires (figure 6.1:b) qui correspondent à une certaine zone de terrain dont l'encombrement transversal est égal ou, plus souvent, supérieur à celui du détail considéré.

Dans une représentation non généralisée, simple réduction d'échelle, le signe de cette zone de terrain viendrait se superposer exactement sur la figuration à grande échelle de la ligne considérée: voir la figure 6.2:(c), (d).

Les petites sinuosités, les irrégularités et les dentelures contenues à l'intérieur de la zone seraient reproduites sur ses bords en leur donnant un aspect irrégulier et déchiqueté qui serait soit imperceptible à petite échelle, soit interprété comme une irrégularité de trait résultant de la mauvaise qualité du dessin: voir la figure 6.2.

Les indentations plus importantes, les tracés compliqués et contournés provoqueraient des superpositions de parties voisines d'une même zone ou de zones différentes mais contiguës, se traduisant, sur la carte à petite échelle, par une impression d'épaississement du trait ou de juxtaposition de traits, sans signification apparente, voir la figure 6.2:(d).

La schématisation comporte donc la suppression des sinuosités et des angularités de faible amplitude et l'amplification de celles qui sont significatives au regard de l'échelle et en fonction de leur importance relative; c'est, en réalité, un problème de sélection de formes.

La schématisation affecte toutes les échelles, sans exception, car, même à très grande échelle, le point du crayon du topographe ou la précision du grain de pixel couvre une certaine zone plus large que l'emprise réelle du détail dessiné.

Il est évident, d'autre part, qu'une ligne droite ou courbe simple, représentant par exemple un élément du réseau géographique, n'est pas affectée par la schématisation, et une ligne quelconque subit une déformation d'autant plus grande que ses dentelures sont nombreuses et de faible amplitude. Ceci se traduit par des discordances sur les longueurs de tracés de lignes différentes figurant sur une carte identique et par des différence entre les longueurs de tracé d'une même ligne sinueuse suivant l'échelle de la carte.

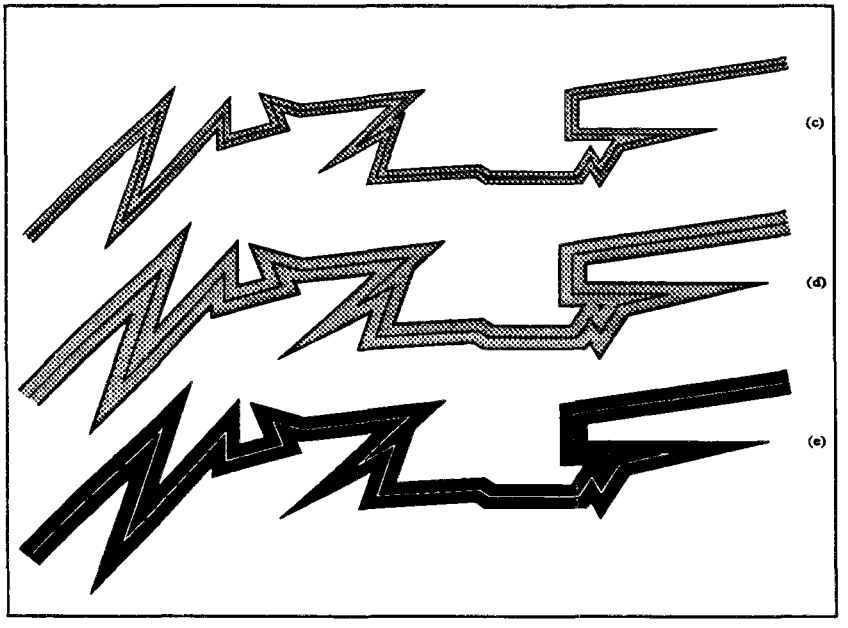


Figure 6.2: Quand l'échelle réduit.

On admet généralement qu'il est souhaitable sinon indispensable de conserver la figuration de certains accidents caractéristiques qui, en toute rigueur, devrait disparaître, en raison de leur dimension, mais qui sont amplifiés, donc déformés, en raison de leur importance locale.

Le choix de ces accidents est nécessairement empirique, il exige du cartographe une grande expérience, des connaissances morphologiques et une vue d'ensemble des divers éléments de la carte puisque l'amplification d'un détail est justifiée par l'existence d'un phénomène différent; enfin, il se traduit par des déformations volontaires mais localisées et non identifiables qui risquent d'être transmises et amplifiées par les schématisations ultérieures jusqu'au moment où l'image cartographique est dénaturée de façon injustifiée.

Donc, une carte à 1/1 000 000 réalisée par généralisation directe à partir de documents de base à 1/100 000 donne souvent une image plus expressive et moins déformée que la même carte dérivant d'un fond identique mais par l'intermédiaire des généralisations successives à 1/200 000 puis 1/500 000.

Néanmoins, la méthode des dérivations successives étant la seule qui soit à la fois rapide, économique et pratique avant que le temps d'un processus d'automatisation plus complète soit arrivé, il est parfaitement normal et logique de l'utiliser pour les opérateurs humains en prenant des précautions pour éviter un effet caricatural excessif, d'où une grande justification pour un processus de généralisation automatique qui se différencie des humains par sa grande capacité de calcul scientifique quelque soit l'importance de données, ce qu'on va présenter des processus dans les sections suivantes sont en principe adaptées à toutes les réductions des échelles sauf cas spécial.

6.4 Signes linéaires

6.4.1 Définitions des segments, arcs, tronçons

Habituellement, les lignes sont représentées par une succession des points équidistants dont l'équidistance est de l'ordre de l'erreur graphique. Par conséquent, la méthode numérique classique du traitement de ligne est de réduire le nombre de données en ne conservant qu'un point sur n (n étant le rapport des échelles). Le problème est que cette numérisation est redondante; la réduction des données risque d'éliminer des points remarquables et caractéristiques.

Ici, nous supposons qu'une ligne quelconque soit numérisée par des segments droits dont la distance est variable, et la distance minimum définie a priori. Seuls les coordonnées de deux extrémités d'un segment droit sont conservées. Donc une ligne droite est représentée simplement par ses deux extrémités.

Nous essayons d'examiner les procédés automatiques de généralisation de ces lignes et nous verrons si certains d'entre eux peuvent donner satisfaction et se concrétiser par des normes graphiques.

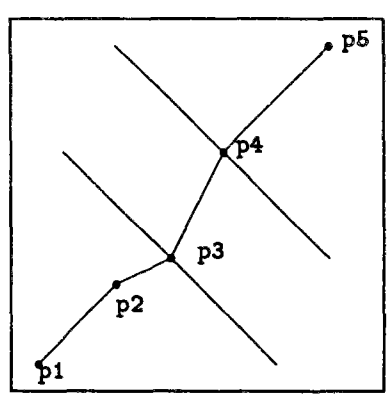


Figure 6.3: Segment, arc, tronçon, route

Pour chaque signe linéaire, on les coupe par des **segments**, des **arcs**, et des **tronçons**. On précise ci-dessous ces notions qui seront utilisées dans la suite de l'exposé.

Définition 1 *Un* **segment** *est une portion de droite définie par ses 2 points extrêmes.*

Définition 2 *Un* **arc** *est une suite de segments sans branches au milieu.*

donc les deux extrémités d'un arc sont forcément reliés avec d'autres arcs.

Définition 3 *Un* **tronçon** *est une suite d'*arcs.

on définit la notion de **route** au sens graphique numérisé:

Définition 4 *Une* **route** *est un tronçon ou une suite d'*arcs.

Sur la figure 6.3, **p1p2**, **p2p3**, **p3p4** sont des segments, **p1p2p3**, **p3p4**, **p4p5** sont des arcs, **p3p4p5**, **p1p2p3p4p5** sont des tronçons, **p1p2p3p4p5** est une route, **p2p3p4** n'est pas un arc, ni **p3p4p5**.

6.4.2 Définition de la convexité

On définit ensuite la notion de **domaine** de **convexité** d'un graphe à trois points. Cela va servir comme base à la fois pour la linéarisation locale d'une ligne, et pour le traitements des carrefours.

Définition 5 *Le* **domaine** *de* **convexité** *d'un tronçon à 3 points est l'intérieur de l'enveloppe convexe de ces 3 points.*

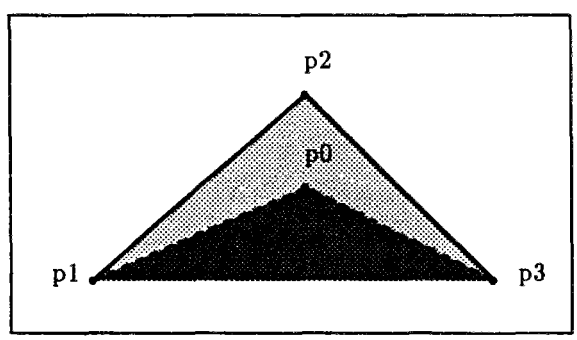


Figure 6.4: Enveloppe convexe de 3 points

Sur la figure 6.4, l'enveloppe convexe de 3 points: p1, p2, et p3 est l'intérieur du triangle (p1p2p3), cet enveloppe convexe est donc le domaine de convexité de ces 3 points.

On définit ensuite une notion de **domaine** de **convexité** d'un graphe linéaire continu.

Définition 6 *Le domaine de convexité d'un graphe linéaire continu est le domaine formé par le graphe linéaire lui même plus, si cette ligne n'est pas encore fermée à ses extrémités, le segment de droite qui relie les deux points extrêmes.*

Le domaine de convexité de 3 points: p1, p2, p3 sur la figure 6.4 est aussi le domaine de convexité de la ligne p1-p2-p3.

Une propriété évidente:

Propriété 1 *Le domaine de convexité d'un graphe à trois points est toujours convexe.*

La propriété suivante décrit l'évolution d'un domaine de convexité.

Propriété 2 *Quand l'un des 3 points se déplace dans leur domaine de convexité, ce domaine rétrécit.*

Sur la figure 6.4, quand le point p2 s'est déplacé au point p0 qui est à l'intérieur du domaine (p1p2p3p1), le domain de convexité est devenu (p1p0p3p1) qui est plus petit que le domaine (p1p2p3p1).

Définition 7 *Quand le domaine de convexité d'un graphe rétrécit, on dit qu'il devient* **moins convexe**. *Quand le domaine de convexité d'un graphe s'agrandit, on dit qu'il devient* **plus convexe**.

Définition 8 *Pour que la convexité d'une ligne continue soit gardée après la généralisation, il faut que:*

- *la base du domaine de convexité reste la même,*
- *le domaine de convexité après la généralisation soit inclu dans le domaine de convexité d'origine.*

Propriété 3 *Si une ligne est composée de trois points dans l'ordre: p1, p2, et p3, seul le déplacement, dans le domaine de convexité de cette ligne, du point du milieu, c'est-à-dire du point p2, garde la convexité de cette ligne.*

Quand le domaine de convexité d'un graphe linéaire devient moins convexe, cette ligne devient moins anguleuse.

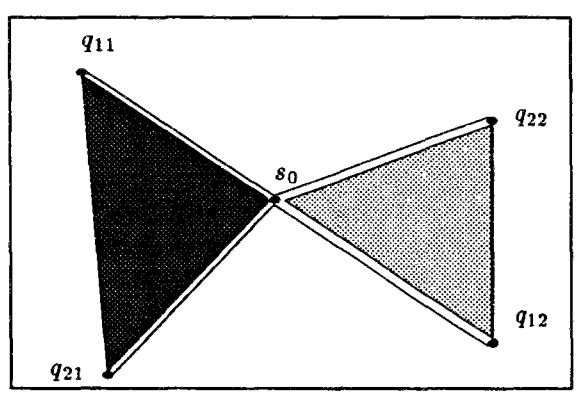


Figure 6.5: Les 2 domaines de convexité de 2 tronçons

Nous supposons dans cet exposé que la convexité d'un arc ou d'un tronçon soit toujours par rapport à leurs deux points extrêmes.

6.4.3 Règles globales de schématisation

On définit des *règles* importantes de linéarisation:

Règle 1 *Un* **arc** *ou* **tronçon** *rectiligne doit rester toujours rectiligne.*

Règle 2 *Un* **arc** *ou* **tronçon** *convexe peut devenir plus convexe dans la limite autorisée ou tout droit, mais jamais concave.*

Règle 3 *Un arc ou un* **tronçon** *concave peut devenir plus concave dans la limite autorisée ou tout droit, mais jamais convexe.*

La règle suivante concerne la relation entre un objet ponctuel et un objet linéaire qui ont une relation de proximité.

Règle 4 *Un objet ponctuel doit toujours garder sa position relative par rapport à un graphe linéaire voisin. C'est à dire que la généralisation du graphe doit conserver la propriété pour ce point d'être intérieur ou extérieur au domaine de convexité de ce graphe.*

Ainsi, un bâtiment qui se trouve d'un certain côté d'une route ne risquera pas de se retrouver de l'autre côté de cette route après généralisation.

Toutes les opérations de généralisation sont strictement soumises à ces règles.

Pour savoir la position d'un point par rapport à un graphe linéaire, rappelons nous le théorème de Cauchy: Soit C le graphe linéaire, S le segment des 2 extrémités de C, $\bar{C} = C + S$, D le domaine ouvert formé par $\bar{C}$, alors on a:

$$\oint_C d\theta = \begin{cases} 2\pi & \text{si le point } P \text{ est dans } D \\ 0 & \text{sinon} \end{cases} \tag{6.11}$$

Dans notre graphe linéaire segmenté fini, les algorithmes seront plus aisément programmés.

6.4.4 Deux segments non alignés: virage

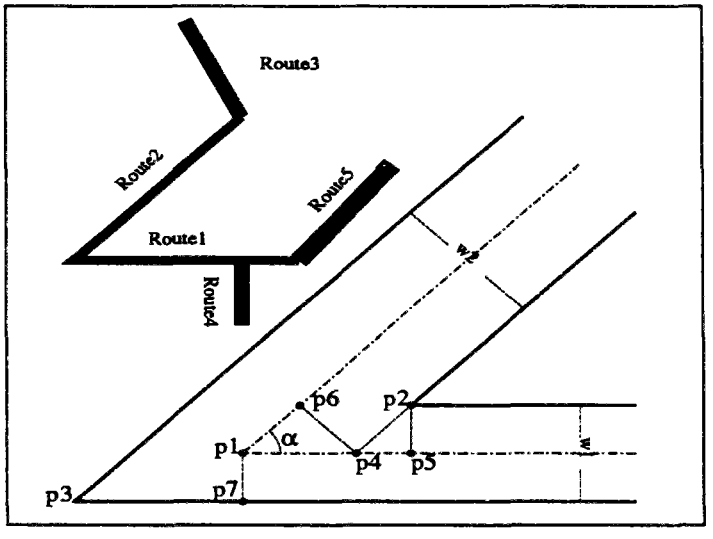


Figure 6.6: 2 segments se croisent

Comme dans la section 2.5 du problème de signe conventionnel, deux routes ont chacune une certaine largeur, (supposons que les traits de routes soient noirs, voir la `route1` et `route2` de la figure 6.6 de la page 104) quand elles se rejoignent, forment un domaine noir d'une certaine taille.

Selon leur largeur et la position de chacun, la taille de ce domaine sera différente aussi. sur la même figure: la **route1** de largeur w_1, la **route2** de w_2, leurs axes se rejoignent au point p1, les axes forment un angle α entre le point p_1 et p_2, il y a bien une longue distance.

p_1 est appelé le *point de croisement d'axes*, p_2 le *point de croisement intérieur* et p_3 le *point de croisement extérieur*.

La distance entre p_1 et p_5 est en effet la projection de largeur de la **route2** sur l'axe de la **route1**, on appelle le segment p_1 et p_5 de cette projection comme *segment de décalage*, sa longueur $w_2w_1(\alpha)$ comme *distance de décalage* sur l'axe de la **route1**. On trouvra également l'emploi de ces notions dans la section 6.6.2 pour le traitement de carrefour.

On essaye d'établir la relation entre la distance de décalage et les largeurs de deux routes et leur angle de croisement. Sur la figure 6.6 on a:

$$\overline{p2p5} = \frac{w_1}{2} \tag{6.12}$$

$$\overline{p4p6} = \frac{w_2}{2} \tag{6.13}$$

$$\overline{p1p4} = \frac{\overline{p4p6}}{\sin\alpha} = \frac{w_2}{2\sin\alpha} \tag{6.14}$$

$$\overline{p4p5} = \frac{\overline{p2p5}}{\tan\alpha} = \frac{w_2}{2\tan\alpha} \tag{6.15}$$

$$\overline{p1p5} = \overline{p1p4} + \overline{p4p5} \tag{6.16}$$

donc, on aura généralement:

$$w_2w_1(\alpha) = \overline{p1p5} = \begin{cases} 0 & \text{si} \begin{pmatrix} \alpha = 0 \\ \text{ou} \quad \alpha = \pm\pi \\ \text{ou} \quad \begin{pmatrix} \pi > \alpha > \pi/2 \\ w_2 < w_1 \end{pmatrix} \end{pmatrix} \\ w_2/2 & \text{si} \quad \alpha = \pi/2 \\ w_1/2\tan\alpha & \text{si} \begin{pmatrix} w_1 < w_2 \\ \pi/2 < \arcsin(w_1/w_2) < \pi \\ \arcsin(w_1/w_2) < \alpha < \pi \end{pmatrix} \\ w_1/2\tan\alpha + w_2/2\sin\alpha & \text{sinon} \end{cases} \tag{6.17}$$

On peut voir que si $w_2w_1(\alpha) > 0$, la projection sera sur l'axe de la **route1** normalement.

Si

$$w_2 < w_1 \tag{6.18}$$

$$\frac{\pi}{2} < \arccos(-w_2/w_1) < \alpha \tag{6.19}$$

on aura: $w_2 w_1(\alpha) < 0$

c'est-à-dire que la projection de la **route2** de largeur w_2 sur la **route1** de largeur w_1 se trouve sur la prolongation de l'axe de la **route1**.

La distance diagonale entre p_1 et p_2 est:

$$\overline{p1p2} = \begin{cases} \textit{trivial} & \text{si} \begin{cases} \text{ou} & \alpha = 0 \\ \text{ou} & \alpha = \pm\pi \\ \text{ou} & \begin{pmatrix} \pi/2 < \alpha < \pi \\ w_2 < w_1 \end{pmatrix} \end{cases} \\ \frac{1}{2}\sqrt{w_1^2 + w_2^2} & \text{si} \quad \alpha = \pi/2 \\ w_1/2\sin\alpha & \text{si} \begin{cases} w_2 > w_1 \\ \pi/2 < \arcsin(w_1/w_2) < \pi \\ \arcsin(w_1/w_2) < \alpha < \pi \end{cases} \\ \begin{pmatrix} = \frac{\sqrt{\overline{p1p5}^2 + \overline{p2p5}^2}}{\frac{1}{2\sin\alpha}\sqrt{w_1^2 + 2\cos\alpha w_1 w_2 + w_2^2}} \end{pmatrix} & \text{si} \begin{cases} \alpha \neq 0 \\ \alpha \neq \pm\pi \\ \begin{pmatrix} \pi \not> \alpha \not> \pi/2 \\ w_2 \not< w_1 \end{pmatrix} \end{cases} \end{cases} \tag{6.20}$$

(6.21)

De l'équation 6.21, on voit que quand les deux segments sont connectés par un point *miter*, et quand l'angle entre ces deux segments α devient trop petit: $\alpha \longrightarrow 0$, la distance diagonale entre p_1 et p_2 : $\overline{p1p2}$ s'accroît de façon inversement proportionnelle avec $\sin\alpha$, ce qui entraine un grand pic autour du point de croisement d'axes et que le hauteur de ce pic ($\overline{p1p2}$) devient trop importante.

Egalement, on a:

$$\overline{p3p1} = \overline{p1p2} \tag{6.22}$$

p_3 est le point du sommet du pic de rencontre de deux segments, plus α est petit, plus ce pic est pointu.

Pour résoudre ce problème de grand pic, du coté du point p_3, au *point de croisement extérieur*, c'est à dire à la rencontre de deux bords extérieurs, on pourra arrondir par un demi cercle de diamètre égal à la largeur de deux segments dans le cas où les deux segments sont de même largeurs, la moyenne de ces largeurs dans le cas contraire.

Tandis que pour le coté intérieur (voisinage du point p_2), on le laissera tel qu'il est.

Par cet arrondisement de la zone de jonction de deux segments, les routes de la figure 6.2 de la page 100 deviennent comme sur la figure 6.7 de page 107.

6.4.5 Linéarisation locale

Cas de petits segments isolés

Soit E_i l'échelle initiale, E_f l'échelle dérivée finale, D_{cv} la largeur du signe conventionnel d'échelle E_f, Voir la figure 6.8 : $D_{\overline{p_2p_3}}$ la longueur du segment $\overline{p_2p_3}$, Si

$$|\alpha_1 - 180^o| \quad < \quad 10^o \tag{6.23}$$

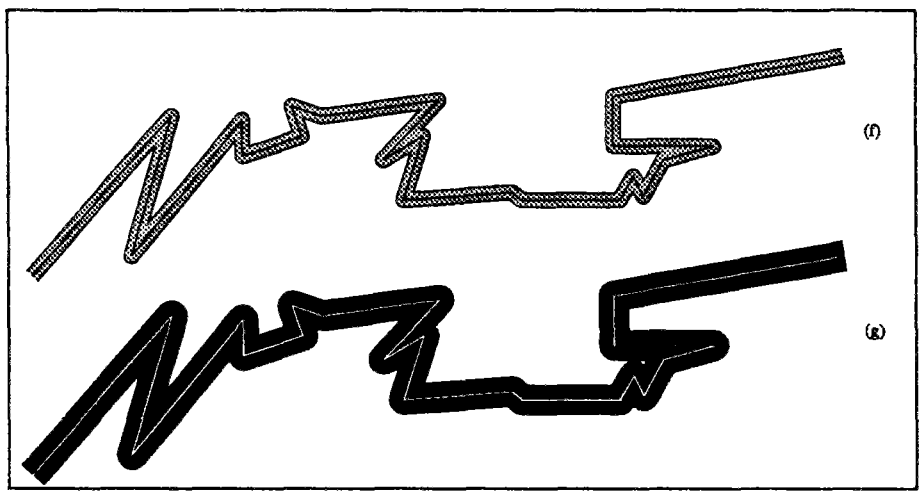


Figure 6.7: Arrondir les coins des segments.

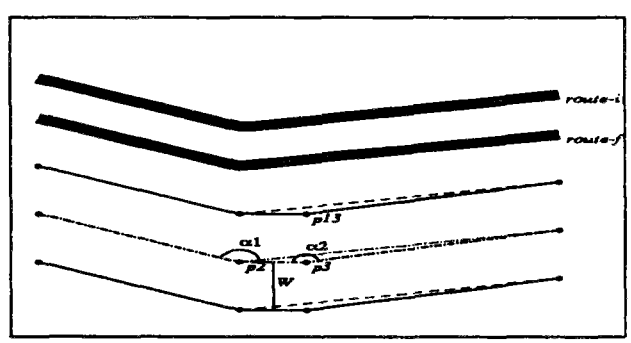


Figure 6.8: Petits segments à supprimer.

$$|\alpha_2 - 180°| < 10° \tag{6.24}$$

$$D_{\overline{p_2p_3}} < .6\frac{E_f}{E_i}mm \tag{6.25}$$

$$< D_{cv}\frac{E_f}{E_i} \tag{6.26}$$

alors le segment $\overline{p_2p_3}$ sera invisible ou confondu avec ses voisins, est sera donc supprimé.

Si

$$|\alpha_1 - 180°| > |\alpha_2 - 180°|$$

alors c'est le point p_3 qui sera enlevé, sinon ce sera p_2.

Cas de petits segments reliés

Si au moins 2 petits segments sont reliés ensemble, ils forment assez souvent de petites sinuosités. Quand le largeur de chaque route augmente, ces petits segments seront peut-être invisibles; sur la figure 6.9, on présente un tel cas. On essaye d'établir le seuil de visibilité pour un tel cas.

Soit la base b_1, coté d_1, sa largeur w, d'après la section 6.4.4, on a:

$$\begin{aligned}
\overline{p_1p_2} &= \frac{w}{2}\left(\frac{1}{\tan\beta} + \frac{1}{\sin\beta}\right) \quad \forall\beta < \pi/2 \\
&= \frac{w}{2\tan\frac{\beta}{2}} && (6.27)\\
d_3 &= \frac{w}{2\tan\alpha} && (6.28)\\
b_3 &= \frac{w}{2\sin\alpha} && (6.29)\\
b_4 &= \frac{w}{2\sin\gamma} && (6.30)
\end{aligned}$$

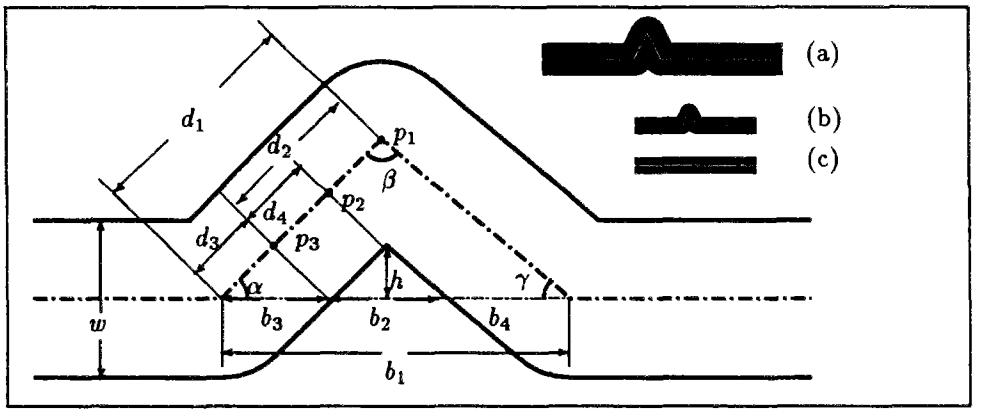


Figure 6.9: Petites sinuosités pouvant être supprimées.

soit h le hauteur, b_2 la base après la déduction de largeur de la route; on a:

$$\begin{aligned}
\overline{p_2p_3} &= d_1 - d_3 - \overline{p_1p_2} \\
&= d_1 - \frac{w}{2\tan\alpha} - \frac{w}{2\tan\frac{\beta}{2}} \\
&= d_1 - \frac{w}{2}\left(\frac{1}{\tan\alpha} + \frac{1}{\tan\frac{\beta}{2}}\right) \quad \forall\beta < \pi/2 && (6.31)\\
b_2 &= b_1 - b_3 - b_4 \\
&= b_1 - \frac{w}{2\sin\alpha} - \frac{w}{2\sin\gamma} && (6.32)\\
h &= \overline{p_2p_3}\sin\alpha && (6.33)
\end{aligned}$$

Pour que cette sinuosité soit visible à l'échelle E_f, il faut que la base et la hauteur soient suffisamment grandes à l'échelle initiale E_i:

$$b_2 > .6mm\frac{E_f}{E_i} \tag{6.34}$$

$$h > 0.4mm\frac{E_f}{E_i} \tag{6.35}$$

A cause de la largeur de route, si la base b_2: après déduction de la partie mangée par la largeur de route, est d'environ moins de $.6mm\frac{E_f}{E_i}$ et leur amplitude moins de $0.4mm\frac{E_f}{E_i}$, ces sinuosités seront invisibles, donc ces petits segments devront être enlevés.

Supposons que cette sinuosité soit régulière: $\alpha = \gamma = \frac{\pi-\beta}{2}$, on aura :

$$\begin{aligned} b_2 &= b_1 - \frac{w}{\sin\frac{\pi-\beta}{2}} - \frac{w}{\sin\frac{\pi-\beta}{2}} \\ &= b_1 - \frac{w}{\cos\frac{\beta}{2}} \end{aligned} \tag{6.36}$$

$$\begin{aligned} h &= \overline{p_2p_3}\sin\alpha \\ &= \cos\frac{\beta}{2}(d_1 - \frac{w(\tan\frac{\beta}{2} + \frac{1}{\tan\frac{\beta}{2}})}{2}) \\ &= d_1\cos\frac{\beta}{2} - \frac{w}{2\sin\frac{\beta}{2}} \end{aligned} \tag{6.37}$$

On peut voir que si la largeur de route devient grande, ou l'angle β devient petit, la base et la hauteur de la sinuosité diminuent jusqu'à être cachés par la largeur de la route.

Lorsqu'on linéarise une ligne anguleuse, on a tendance à supprimer toutes les petites sinuosités. La convexité de cette ligne risque donc d'être trop changée. Les règles globales de généralisation définies dans la section 6.4.3 sont à prendre en compte.

Linéarisation locale et globale

On distingue deux sortes de linéarisations, locale et globale.

Localement, on linéarise arc par arc en supprimant des petites sinuosités, des petits segments selon les règles évoquées précédemment, c'est à dire qu'on ne touche pas aux points de croisement; on parle donc de *linéarisation locale.*

Une ligne représentant une voie de communication n'est jamais seule, elle est toujours reliée avec d'autres lignes par des croisements qu'on appellera des *carrefours.* Pour garder les positions relatives de ces carrefours par lesquels une ligne est liée avec ses voisins, on a aussi des linéarisations à faire quand on fait translater cette ligne de façon globale. On sera donc obligé de déplacer le point de croisement au moment de la linéarisation; on parlera de *linéarisation globale.* On doit donc avoir une vue plus générale pour l'ensemble des graphes.

6.5 Règles d'harmonisation

On a cité des règles de généralisation adaptées à des éléments séparés et indépendants par leur nature, leur implantation, leur forme.

Dans la réalité, au fur et à mesure que l'on s'éloigne de la structure, de l'implantation et du nombre exact des objets, on accroît la part de convention et on risque, consciemment ou non, de rompre l'équilibre naturel de la carte en favorisant certaines catégories de détails au détriment des autres; alors que tous les éléments graphiques de la carte sont étroitement solidaires les uns des autres, leur sélection et leur schématisation doivent être équilibrées et coordonnées de manière à conserver au mieux les proportions relationnelles bien que certains détails privilégiés conservent leur position exacte et que les autres soient décalés.

Ces conflits et ces décalages sont la conséquence inévitable de l'emploi de signes conventionnels et de la schématisation; au tableau 2.3 sur la page 28, une autoroute de 28 m sera représentée par un signe de .7mm sur une carte d'échelle 1:1000 000, soit 700 m à l'échelle de la réalité, ce qui donne une amplification à peu près de 25 fois de la réalité.

Il existe deux sortes de conflits:

- *local*: ceci est dû aux sinuosités de lignes avec l'emploi de signe conventionnel. Il peut être détecté et résolu par la linéarisation local en supprimant les segments en question. (voir la section 6.4.5).
- *global*: ceci est dû aux superpositions et intersections entre les objets alors qu'ils devraient être écartés. Cette sorte de conflit demande une vue d'ensemble de la région ou de toute la carte.

En résumé, tout conflit vient d'un problème d'espace: il manque toujours de la place.

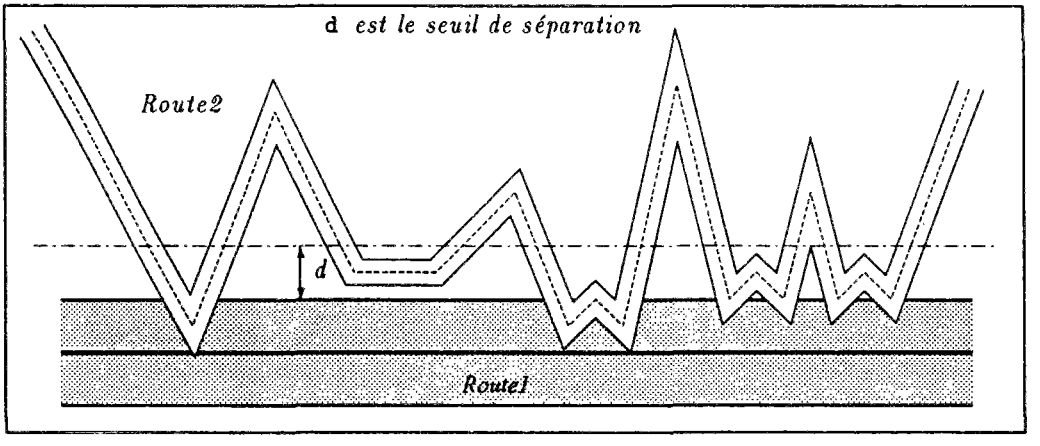


Figure 6.10: Superposition de deux signes linéaires

Les décalages affectent non seulement les signes contigus mais également, dans les régions de planimétrie dense, les éléments situés à une certaine distance. Par

exemple le signe d'une route repousse de chaque coté les détails qui la bordent, sur la figure 6.10 de la page 110, les axes de *route2* et *route1* n'étaient pas superposés l'un et l'autre, mais le largeur de chaque signe conventionnel fait que les routes s'intersectent, en plus, une distance de **d** qui est **seuil** de séparation des lignes s'ajoute entre les 2 routes, une des deux ne peut plus garder sa propre position, ou bien encore l'une devrait être supprimée si elle n'est pas d'une grande importance, pour une plus grande clarté du signe conventionnel de l'autre.

Selon la règle d'existence 1 de la page 93, on doit faire en sorte de garder tout ce qu'on peut; donc un premier essai de décalage est obligatoire, ensuite interviendra un essai de suppression.

La généralisation aboutit ainsi à une égalisation apparente de phénomènes de dimensions réelles très différentes, égalisation qui fausse l'appréciation visuelle des rapports de surfaces.

Les problèmes se posent: lequel décaler? lequel supprimer? Cela dépend de l'importance de chacun.

6.5.1 Hiérarchie des éléments

Les règles d'harmonisation ont pour base l'adoption d'une hiérarchie préférentielle des graphismes en fonction de leur nature, certains restent en place, d'autres étant décalés. Les relations d'ordre devraient être définies selon la destination et le thème de la carte puisque la nature des détails primordiaux dépend de ces critères essentiels.

En général, la liste de préférence sera:

- ligne hydrographique qui est presque toujours à sa position exacte; son tracé a une influence prépondérante sur l'emplacement, l'agencement relatif et la forme de voies de communication.
- ligne de cote et rive de lac.
- voies ferrées qui sont assez souvent rigides.
- voies routières selon un classement par appartenance administrative:
  - autoroute
  - route nationale
  - route départementale
  - voie communale
  - voie urbaine

  auquel s'ajoute d'autres statuts administratifs:
  - voie express
  - route internationale (européenne)
  - route à grande circulation

  un classement selon la nature de leurs voies, par exemple à 1/25 000:

  1. Autoroute

2. Route Express
3. Route à chaussées séparées
4. Route à plus de 3 voies
5. Route à 2 voies larges
6. Route à 2 voies étroites
7. Route moins de 5 m
8. Autre route étroite régulièrement entretenues.
9. Autre route étroite non régulièrement entretenues.
10. Route irrégulière.
11. Ruelle d'agglomération.
12. Chemin d'exploitation
13. Sentier
14. chemin de terre ou privé.

- Construction
- divers clôtures
- les limites de végétation et de culture
- les limites administratives

Selon cette hiérarchie, on effectue des opérations d'harmonisation.

Règle-d'harmonisation 1 *Les décalages s'effectuent selon cette hiérarchie.* ♣

Règle-d'harmonisation 2 *Lorsque des signes équivalents sont contigus et que cette proximité impose un aménagement, on fait en sorte que soit maintenu en place le centre ou l'axe du groupe qu'ils forment.* ♣

Les règles globales citées dans la section 6.4.3 sont aussi des règles d'harmonisation.

Les algorithmes proposés pour résoudre les conflits entre les éléments sont basés sur les principes:

- Un tronçon de type rectiligne doit rester rectiligne. Quand on déplace un segment (qui est toujours droit par définition), en fait on ne déplace que les deux extrémités de ce segment. Si ce segment constitue un des segments qui forment un tronçon rectiligne, alors le déplacement du segment en question revient à déplacer les deux extrémités du tronçon auquel ce segment appartient. Une priorité de fait est souvent accordée aux voies ferrées dont le tracé rigide, fait de lignes droites et d'arcs de cercles, ne s'accomode pas de sinuosités locales; la même remarque s'applique aux autoroutes alors que les routes ordinaires supportent mieux des décalages limités constituant des entorses locales à leur tracé réel.

- Un autre principe de l'harmonisation est le maintien de la cohérence: cohérence entre les éléments équivalents ou identiques qui doivent subir une sélection et une schématisation semblable.

  La relation locale entre les éléments doit être maintenu. Parmi un groupe d'éléments, les positions relatives par rapport aux autres doivent rester les mêmes.

- En tout cas, il faut toujours chercher à minimiser les décalages et à conserver au mieux les rapports relatifs de longueurs et de surfaces.

  Les éléments doivent être déplacés d'une distance minimum pour maintenir l'intégrité du groupe. La distance minimum est définie par les distances initiales entre eux et par la règle de lisibilité.

6.5.2 Séparer deux signes voisins ponctuels

En calculant la distance entre deux éléments ponctuels, on peut facilement savoir s'ils ont besoin de subir ou non un traitement pour préserver la distance minimum entre eux.

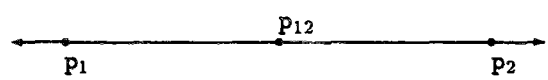


Figure 6.11: Séparation de deux éléments ponctuels

- La distance de déplacement est fonction de leur distance de séparation initiale et de la distance d'écartement minimale définie par la règle de lisibilité (voir la section 2.4).

- La direction est choisie de façon à garder leur relation locale et relative par rapport à un point fixe défini par les positions initiales. Ce point fixe P_{centre} de référence est défini:

  - *quantitativement*: en fonction des positions initiales de deux éléments, par défaut au milieu de l'axe initial de deux éléments, s'ils sont de même catégorie:

    $$\vec{P}_{centre} = \frac{\vec{p}_1 + \vec{p}_2}{2} = \vec{p}_{12} \tag{6.38}$$

    Par exemple le point p_{12} est le milieu des positions de deux points: p_1 et p_2 sur la figure 6.11; dans ce cas, la direction de déplacement est vers la gauche pour le point p_1, vers la droite pour le point p_2.

  - ou *qualitativement*: en fonction de l'importance de chacun:
    - sur le point qui est dans l'absolu plus important que l'autre selon la hiérarchie des objets prédéfinis.
    - si on n'a pas de choix, sur le point dont la direction pour écarter de l'autre point n'est pas disponible.

- ou par un compromis, sur un point entre les deux positions en fonction de coefficient d'importance de chacun.

$$\vec{P}_{centre} = k\vec{p}_1 + (1-k)\vec{p}_2 \tag{6.39}$$

Le coefficient $k \in [0,1]$ représentent l'importance relative entre les deux éléments.

Si la distance entre ces deux éléments doit être de D, $D > |\vec{p}_1\vec{p}_2|$, les deux éléments doivent être déplacés à leurs nouvelles positions p_{11}, p_{22} : $p_1 \rightarrow p_{11}$ et $p_1 \rightarrow p_{22}$ de telle façon que leur centre fixe $\vec{P}_{centre}$ soit maintenu:

$$|\vec{p}_{11}\vec{P}_{centre}| = (1-k)D \tag{6.40}$$
$$|\vec{p}_{22}\vec{P}_{centre}| = kD \tag{6.41}$$

En attribuant correctement le coefficient k, on peut donc contrôler la direction et la distance de déplacement de ces deux points. Par exemple sur le point p_1 s'il est plus important que le point p_2, et que la direction de la droite de p_2 soit disponible; dans ce cas, $k = 1$, le point p_1 reste sur son ancienne position, le point p_2 doit se déplacer vers la droite.

6.5.3 Séparer un groupe de signes ponctuels

Déplacer séparément les éléments entraîne fatalement la propagation de ces déplacements. On cherche donc plutôt des actions ensemblistes que des actions individuelles. Mackaness et Fisher[21] présente un algorithme d'identification et résolution de conflits entre un groupe d'éléments ponctuels, avec la même idée que l'équation 6.38 en cherchant un centre de référence quantitative. Mais ils n'ont pas pris en compte les références qualitatives des éléments à déplacer. Les éléments de ce groupe sont implicitement tous de même type:

$$\vec{P}_{centre} = \frac{\vec{p}_1 + \vec{p}_2 + \cdots + \vec{p}_n}{n} \tag{6.42}$$

Le point P_{centre} sera vers l'endroit où les éléments sont denses. L'intérêt de cet algorithme est de mettre l'accent sur un ensemble d'objets et le contraste au groupe, et non sur les

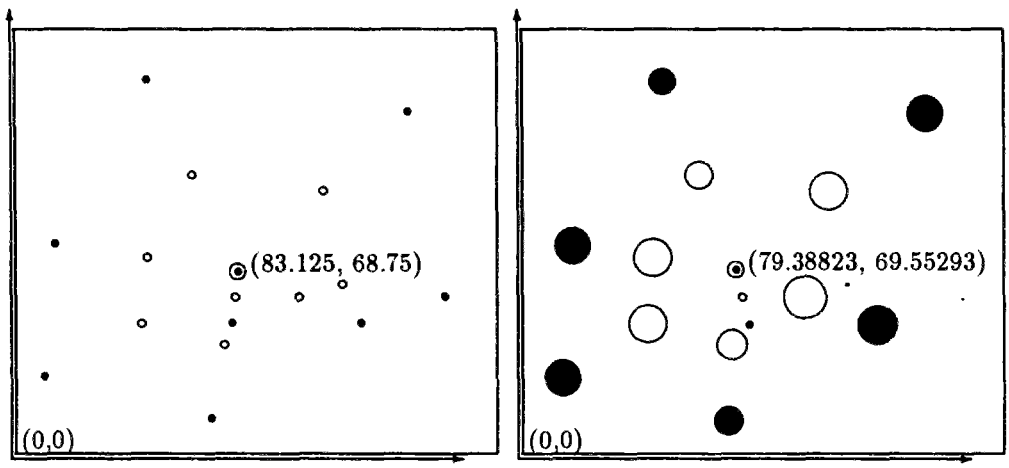
On améliorera l'équation 6.42 de l'algorithme de Mackaness et Fisher en introduisant des coefficients d'importance $m_1, \cdots, m_n$ qui joueront le rôle des masses dans la définition du barycentre:

$$\vec{P}_{barycentre} = \frac{m_1\vec{p}_1 + m_2\vec{p}_2 + \cdots + m_n\vec{p}_n}{m_1 + m_2 + \cdots + m_n} \tag{6.43}$$

Ce barycentre sera donc vers l'endroit où les éléments sont les plus denses et aussi où les objets sont les plus importants.

En cas de déplacement nécessaire, les distances des éléments à déplacer seront calculée par une homothétie de rapport k par rapport à ce barycentre:

$$\begin{aligned} \vec{p}_{id} &= \vec{P}_{barycentre} + k(\vec{p}_i - \vec{P}_{barycentre}) \\ &= (1-k)\vec{P}_{barycentre} + k\vec{p}_i \end{aligned} \tag{6.44}$$



barycentre ° position initiale • position finale $d = m_i$

Figure 6.12: Expansion de 2 sans prendre en compte l'importance des éléments par rapport à leur point du centre: $\vec{P}_{centre}$

Figure 6.13: Expansion de 2 en prenant en compte l'importance de chaque élément par rapport à leur barycentre $\vec{P}_{barycentre}$

La distance de déplacement d'un point est proportionnelle à la distance initiale entre le point et ce barycentre.

Les figures 6.14, 6.12, 6.13 montrent des exemples des déplacements par l'expansion, soit par rapport au centre de référence ordinaire des éléments sans considérer leur importance (sur les figures 6.14a, 6.12, tous les éléments sont représentés par des points de la même taille), soit par rapport à leur barycentre en prenant en compte les importances (qui sont représentées par le diamètre du rond sur les figures 6.14b,

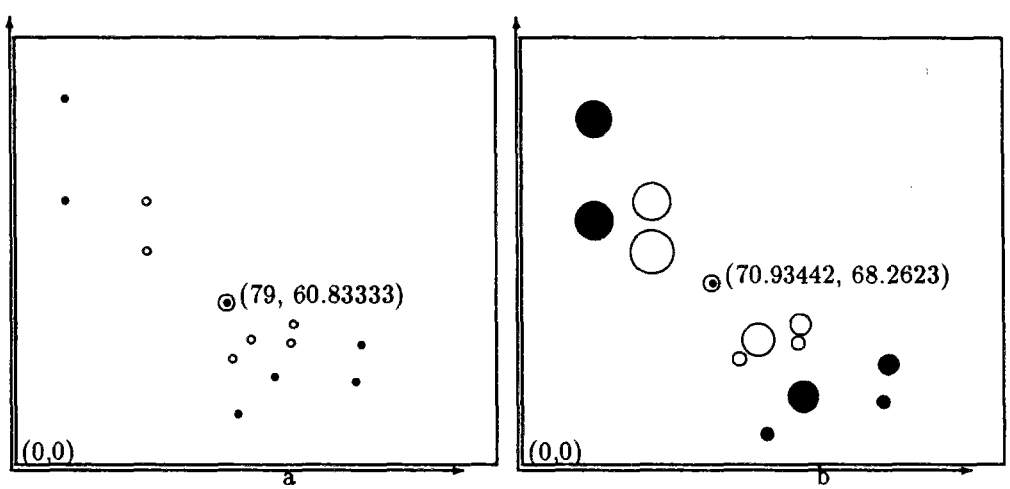


Figure 6.14: Homothétie d'un groupe d'éléments ponctuels

6.13) de chaque élément à déplacer. Les 2 groupes d'éléments ponctuels sont produits aléatoirement par une fonction **Lelisp**.

L'avantage est que le centre de référence d'un groupe d'objets est maintenu sur place, et que les objets les plus importants sont les moins déplacés, la majorité des objets autour du barycentre sont également les moins déplacés, ainsi est minimisé l'ensemble des décalages des éléments. La forme de ce groupe d'éléments est également relativement conservée.

Mais l'homothétie d'un groupe d'éléments par rapport à leur barycentre fait perdre partiellement la relation locale entre ce groupe et le reste de la carte. Une méthode est de remplacer le facteur constant de déplacement k dans l'équation 6.44 par une fonction $f(x)$ de distribution qui varie en fonction de la distance x entre le point et le barycentre.

$$f(x) = 1 + (k-1)\exp^{-\frac{x^2}{2}} \tag{6.45}$$

on a:

$$f(0) = k$$

k est une constante.

$$\lim_{x\to\infty} f(x) = 1$$

Pour comparer avec l'expansion sur figure 6.13, la figure 6.15 donne une autre expansion ($k = 2$) en utilisant les équations 6.46, 6.45, avec

$$x = \frac{d_{p_c p_1}}{\mathrm{d}_m}$$

$$\mathrm{d}_m = \frac{m_1 d_{p_c p_1} + m_2 d_{p_c p_2} + \cdots + m_n d_{p_c p_n}}{m_1 + m_2 + \cdots + m_n} \tag{6.46}$$

où p_c est le barycentre $\mathrm{P}_{barycentre}$, $d_{p_c p_n}$ est la distance entre le barycentre p_c et le point p_n.

Ainsi est mieux résolu le problème de déplacement d'un groupe d'éléments en évitant la propagation du déplacement des autres objets introduit par le déplacement d'un seul objet. Le problème de contraste de relativité entre les objets est également résolu en cas de conflit. Mais en aucun cas on ne considère les positions relatives entre les éléments. Une méthode est d'attribuer les coefficients de façon différente pour rendre compte de relations entre eux.

En changeant le coefficient d'importance m_i des éléments, le barycentre sera donc déplacé pour répondre à un besoin particulier. Par exemple, si on estime qu'un élément p_i est absolument plus important que les autres, et que son déplacement est strictement interdit, alors on peut lui attribuer l'importance: $m_i = 1$, et 0 aux autres. D'après l'équation 6.43, on aura:

$$\vec{\mathrm{P}}_{centre} = \vec{\mathrm{p}}_i$$

Le barycentre coïncidera donc avec cet élément qui ne sera déplacé en aucun cas.

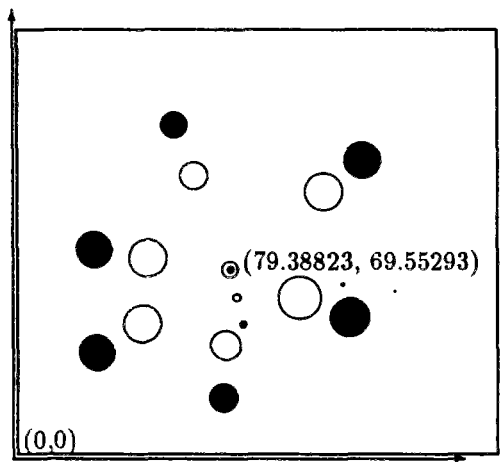


Figure 6.15: Homothétie avec une distribution exponentielle

6.5.4 Séparer deux signes voisins linéaires

On a défini dans la section 2.4 le **seuil de séparation**, qui est l'écart minimal nécessaire entre deux éléments graphiques voisins pour les isoler à l'oeil nu et qui correspond au pouvoir séparateur de l'oeil. Enfin, il ne suffit pas de distinguer et de séparer les éléments graphiques, il faut encore apprécier les différences de dimension ou de valeur qui expriment des paliers distincts en classement quantitatif (ou simplement ordonné). Sur la figure 6.10 de la page 110, la *route2* s'intersecte avec la *route1*; par la règle d'importance, c'est à la *route2* de s'éloigner de la *route1*. Plusieurs décisions pourraient être prises:

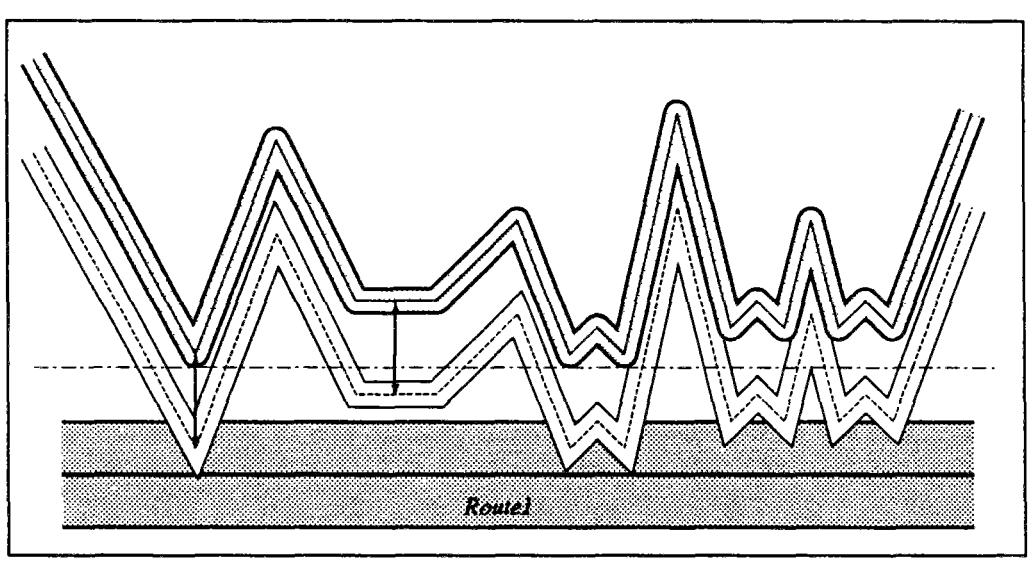


Figure 6.16: Translation globale sans contrainte

1. S'il y a de la place à l'autre coté de la route, on décale entièrement la *route2* (ou plutôt le segment concerné de cette *route2* qui n'a pas d'intersection avec d'autre routes), voir la figure 6.16.
2. S'il y a des contraintes, on ne décale que les segments qui sont dans la zone d'intersection, figure 6.17.
3. Pour garder localement la convexité des arcs ou tronçons , une linéarisation locale est éventuellement nécessaire.

En fait,la séparation des signes linéaires se ramène aussi à:

- une translation globale si c'est possible, c'est à dire que le coté de la translation est suffisamment libre (figure 6.16).
- une linéarisation locale ou globale (figure 6.17).

6.5.5 Séparer un groupe de signes voisins linéaires

Les principes théoriques de la séparation d'un groupe de signes linéaires sont les mêmes. On met l'accent sur l'intégrité d'un signe linéaire. Il s'agit aussi de prendre un axe de référence (de type barycentre linéaire) que pour les éléments ponctuels. Trouver cet axe de référence n'est pas facile. Une recherche profonde des algorithmes est nécessaire.

6.5.6 Translation d'un signe linéaire

La direction de déplacement d'une ligne est à priori donnée par la demande d'une action telle que **`Eloigner R1 de R2`**. Donc elle est toujours à la direction verticale à la ligne fixe.

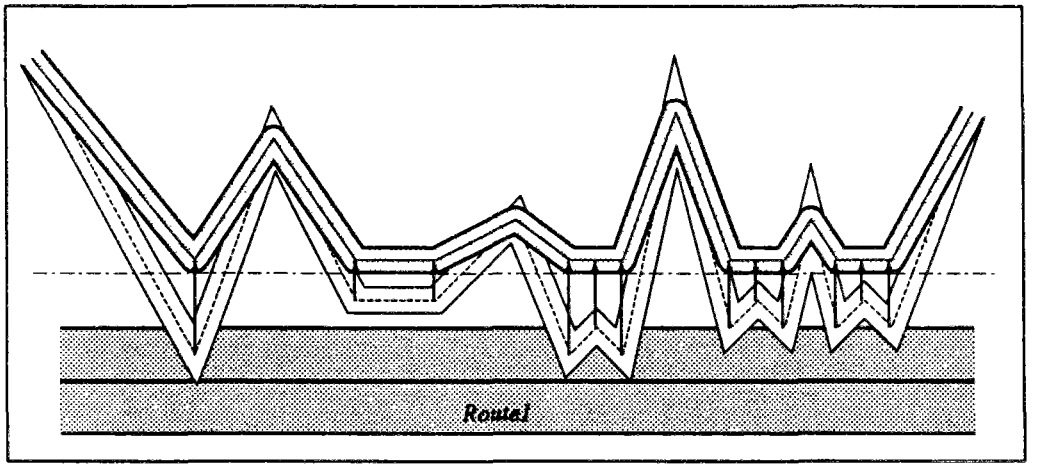


Figure 6.17: Translation locale + linéarisation locale

Définition 9 *Une* **`translation locale`** *est une translation d'un arc.*

Il s'agit d'une translation des points d'un seul arc, les deux extrémités d'un arc qui sont reliés avec d'autres graphes linéaires restent fixes dans cette catégorie de translation;

Définition 10 *Une* **translation globale** *est une translation de plusieurs arcs.*

Il s'agit d'une translation d'ensemble d'un graphe linéaire, c'est -à dire de plusieurs arcs ensemble. Ce dernier problème est discuté dans la section suivante et dans la section 6.6.2 pour les problèmes de carrefour.

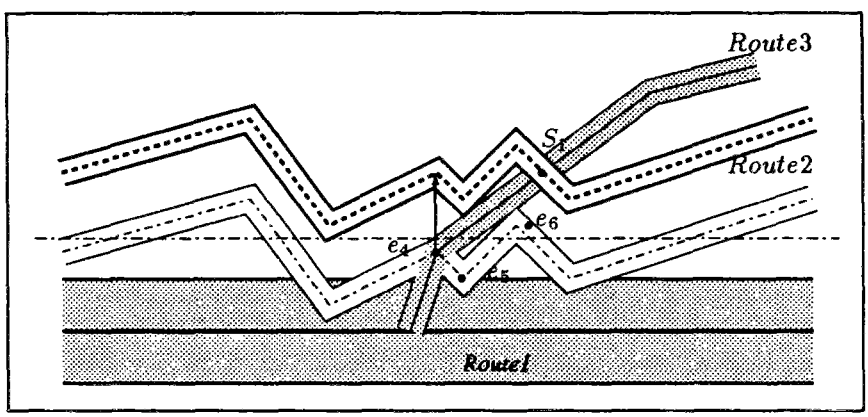


Figure 6.18: Translation avec contraintes: avant linéarisation

Une translation d'un point avec des coordonnées (x, y) est représentée par le changement de coordonnées: de (x, y) à $(x + dx, y + dy)$.

- Pour un point de croisement, si ses coordonnées sont devenues: $(x + dx, y + dy)$, ce point a de forte chance de se trouver sans contact avec ses anciens segments. Par exemple, sur la figure 6.18, e_4 est un point de croisement entre la *Route3* et la *Route2*. La *Route2* est à écarter de la *Route1*. Après une translation de la *Route2*, le nouveau point de croisement entre cette dernière et la *Route3* se trouve sur le point $S1$, et non le point e_4 translaté.

  C'est à dire que le croisement de cette route ne sera pas forcément sur $(x + dx, y + dy)$, la position du nouveau croisement est donc mise en question, c'est la nouvelle position du croisement qui deviendra la position du point en question de cette route, et pas simplement les $(x + dx, y + dy)$. Le calcul de la nouvelle position sera dans la section suivante.

  Le maintien de la position relative du point de croisement avec ses segments liés est cher, car il change plus ou moins la convexité de la route à translater. Des problèmes pour les points de voisinages se posent aussitôt; ces points de voisinages devront toujours garder la convexité de l'arc auquel ils appartiennent malgré toute les conséquences des translations.

- Comme les points de croisement ne sont jamais à l'emplacement défini par la pure translation, cela pose aussi un problème pour les autres points voisins. Dans un premier temps, tous les points sont translatés normalement; les coordonnées (x, y) sont devenus $(x + dx, y + dy)$, qui seront ensuite réexaminés et corrigées en consultant les anciennes convexités des arcs de la route:

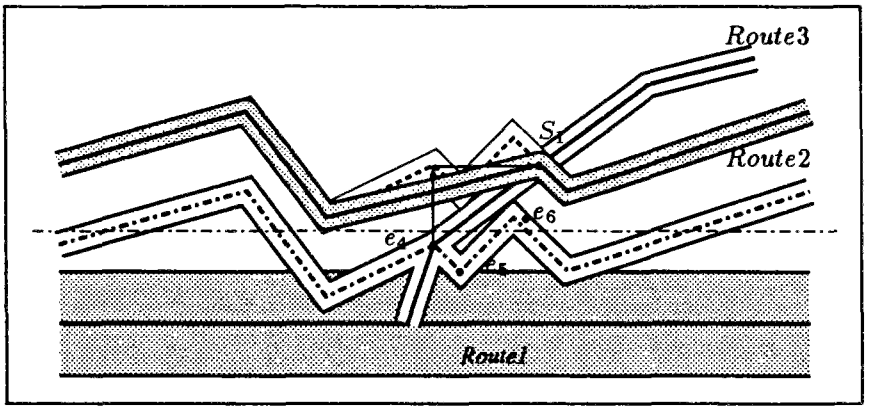


Figure 6.19: Translation avec contraintes: après linéarisation

- si un point est toujours dans l'ancienne convexité de l'arc dont il dépend, il sera gardé tel qu'il a été translaté.
- si un point n'est plus dans l'ancienne convexité de l'arc dont il dépend, il sera ramené sur le segment droit qui est formé par ses deux points de voisinage sur la même route; en effet c'est comme si ce point était éliminé (voir la figure 6.19.

6.6 Traitement des carrefours

On accorde une attention particulière et un traitement spécial aux intersections de détails linéaires appartenant à la même catégorie: croisement, carrefours, bifurcation de routes et de chemins, raccordements de voies navigables ou de voies ferrées, etc, on les appellera **carrefours**.

Plutôt que de chercher à respecter scrupuleusement l'axes des éléments en présence, ce qui défigure fréquemment l'aspect de l'intersection et donne une idée erronée de la position relative des lignes, il est préférable de traduire sans ambiguïté cette disposition relative de lignes, quitte à décaler les lignes secondaire. On donnera des exemples de cette schématisation.

Une fois de plus, s'affirme le caractère conventionnel de la représentation cartographique et se manifeste la difficulté de réaliser un compromis entre la précision, les positions relatives des objets et la lisibilité.

6.6.1 Importance de carrefours

L'importance d'un carrefour est fonction de l'importance des routes qui y arrivent. Les importances dans la hiérarchie des voies de communication sont définie dans la section 6.5.1, sur la page 111 .

Définition 11 *L'indice d'importance d'un carrefour est défini par la somme des valeurs de chacune des routes (selon la hiérarchie définie dans le paragraphe 6.5.1) qui constituent ce carrefour.*

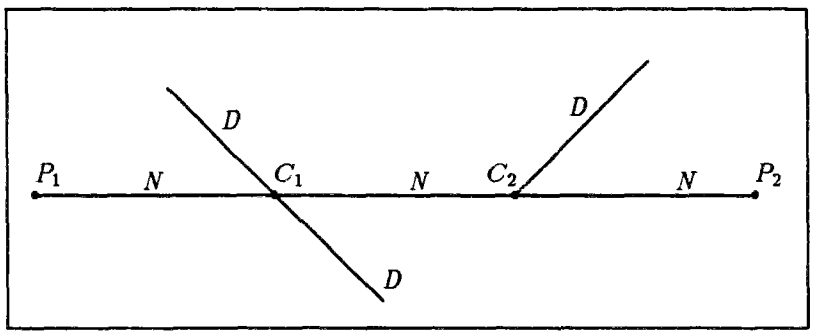


Figure 6.20: Importance de carrefour

C'est cette indice qui permettra de comparer l'importance de deux carrefours. Par exemple, sur la figure 6.20, il y a deux carrefours:

- carrefour1:
  1 route nationale: 2 arcs de type 'N': $\mathbf{n_N^{c1} = 2}$
  1 route départementale: 2 arcs de type 'D': $\mathbf{n_D^{c1} = 2}$
- carrefour2:
  1 route nationale : 2 arcs de type 'N': $\mathbf{n_N^{c2} = 2}$
  1 route départementale qui termine à ce carrefour: 1 arc de type 'D': $\mathbf{n_D^{c2} = 1}$

On aura la comparaison d'importances de ce deux carrefours:

$$\mathbf{n_N^{c1} = n_N^{c2}} \tag{6.47}$$

$$\mathbf{n_D^{c1} > n_D^{c2}} \tag{6.48}$$

donc on a **carrefour1** > **carrefour2**.

6.6.2 Domaine de convexité d'un carrefour

On définit la notion de **carrefour**:

Définition 12 *Un* **carrefour** *est composé d'au moins deux graphes linéaires qui se croisent à un point commun.*
Le **sommet** *de ce carrefour est ce point de croisement.*
Les **arc-carrefours** *sont les segments qui arrivent sur le sommet de carrefour.*

Sur la figure 6.5 de la page 103, la ligne $(q_{11}s_3q_{21})$ et la ligne $(q_{22}s_3q_{12})$ forment un carrefour, elles se croisent au point s_3, donc, s_3 est un **sommet** de **carrefour**, les segments $q_{11}s_3$, $q_{21}s_3$, $q_{12}s_3$, $q_{22}s_3$ sont ses **arc-carrefours**.

On a déjà défini la notion de convexité générale dans la section 6.4.2, on définit maintenant un **domaine de convexité d'un carrefour** ci-dessous:

Définition 13 *Le* **domaine** *de* **convexité** *d'un carrefour est le domaine sur lequel toutes les routes aboutissant à ce carrefour deviennent moins convexes mais gardent toujours leur convexité quand le sommet du carrefour se déplace sur ce domaine.*

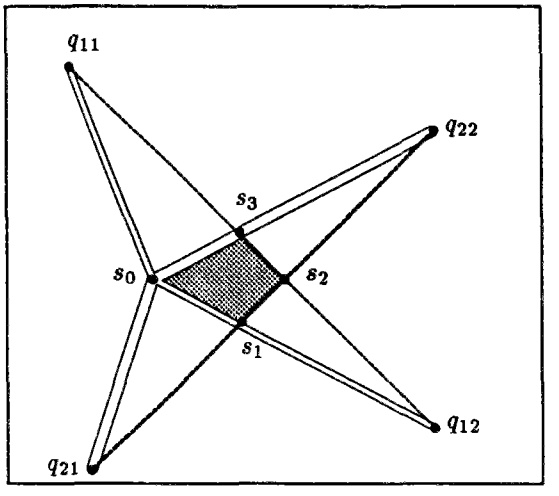


Figure 6.21: Convexité d'un carrefour

Pour mesurer le domaine de convexité d'un carrefour, on examiner les arcs autour du sommet du croisement.

Chaque route définit au voisinage d'un carrefour un triangle constitué par le sommet du carrefour et les extrémités des deux segments voisins, triangle dont l'aire peut être nulle est le domaine de convexité de la route correspondante à ce carrefour. on peut donc avoir la propriété suivante:

Propriété 4 *Le* **domaine de convexité** *d'un carrefour est l'intersection de tous les domaines (triangulaires) de convexité de chaque graphe linéaire qui aboutit à ce carrefour.*

On définit la notion de *sommet de convexité* d'un **carrefour**:

Définition 14 *Le* **sommet** *de* **convexité** *d'un carrefour est un point déterminé par l'intersection des cordes qui sont les premiers segments des arcs aboutissant à ce carrefour.*

A un **sommet de convexité** d'un carrefour, toutes les routes traversent ce point tout droit. Par exemple, sur la figure 6.21 de la page 122, une route(q_{11}, s_0, q_{12}) croise une autre route$(q_{21}, s0, q22)$ au carrefour s_0, le **domaine**$(s_0, s_1, s_2, s_3, s_0)$ est donc un **domaine de convexité** de ce carrefour s_0, le point s_2 est un **sommet de convexité** de ce domaine. Si le point s_0 est déplacé au point s_2, le **sommet de convexité**, les arcs des routes (q_{11}, s_0, q_{12}), $(q_{21}, s0, q22)$ seront tout droits.

On va essayer de voir les domaines de convexité des carrefours en mouvement. Pour un carrefour de 4 segments seulement, cette section sera allégée. Malheureusement, ce n'est pas toujours le cas, mais nous commençons par ce cas qui est relativement plus simple.

Supposons sur la figure 6.21, que le point s_0 se déplace sur son domaine de convexité: les routes deviendront de plus en plus droites mais en gardant toujours leur convexité, jusqu'au point s_2, où les deux routes deviennent complètement droites.

Ce **domaine** de convexité n'existe pas toujours, cela dépend de position de chaque segment qui arrive sur ce point de croisement.

Par exemple, sur la figure 6.5 de la page 103, les 2 routes $R_1(q_{11}, s_3, q_{21})$ et $R_2(q_{12}, s_3, q_{22})$ n'ont pas de **domaine** de convexité, puisque les deux triangles de chaque arc n'ont pas de conjonction. Quand le point s_3 se déplace, au moins une de 2 routes sortira toujours de son domaine de convexité, et donc on dira qu'elle deviendra toujours plus convexe.

4 formes d'existence d'un domaine de convexité sont évoquées:

1. **zonal**, ce carrefour est donc flexible, puisque qu'on peut le faire déplacer si nécessaire dans ce domaine tout en gardant les convexités des routes. Le domaine $(s_0, s_1, s_2, s_3, s_0)$ du carrefour s_0 sur la figure 6.21 de la page 122 est un tel cas.

2. **linéaire**, ce carrefour est aussi flexible sur un segment commun de cette ligne.

   Le carrefour C_2 est flexible sur le segment C_1P_2 sur la figure 6.20 à la page 121. Un glissement de la route (C_2D) sur ce segment est possible pour répondre au besoin de déplacement de ce carrefour.

3. **ponctuel**, c'est à dire que ce domaine ponctuel de convexité est sur le **sommet** de convexité.

   On ne peut faire un simple déplacement du sommet de carrefour pour déplacer ce carrefour sans détruire les convexités des routes, seule une translation globale de toute ou partie de routes aboutissant à ce carrefour est possible. Le carrefour C_1 est presque gelé sur le point C_1 sur la figure 6.20 à la page 121. Une translation de la route(DC_1D) ou de la route($P_1C_1C_2P_2$) ou des deux ensemble est nécessaire en cas de demande de déplacement de ce carrefour.

4. **nul**, sans domaine de convexité, un déplacement de ce carrefour éloignera toujours au moins une des routes de son propre domaine de convexité. On choisira donc de déplacer une des routes dont la priorité est moins grande pour favoriser le déplacement des autres routes.

   Tel est le cas sur la figure 6.5 de la page 103.

Calcul du sommet de convexité

On reprend la figure 6.21 de la page 122. Supposons qu'une droite se représente par deux points dont les deux coordonnées sont (x, y), et que les coordonnées de chaque points sont les suivants:

$$\vec{q}_{11}(x_{11}, y_{11}),\ \vec{q}_{12}(x_{12}, y_{12}),\ \vec{q}_{21}(x_{21}, y_{21}),\ \vec{q}_{22}(x_{22}, y_{22})$$

Le **sommet** de convexité $\vec{s}_2(x, y)$ est sur:

$$\vec{s}_2 = \vec{q}_{11} + t_1(\vec{q}_{12} - \vec{q}_{11}) \quad \text{pour } 0 \leq t_1 \leq 1 \tag{6.49}$$

$$\vec{s}_2 = \vec{q}_{21} + t_2(\vec{q}_{22} - \vec{q}_{21}) \quad \text{pour } 0 \leq t_2 \leq 1 \tag{6.50}$$

de l'équation 6.49 et 6.50, on a

$$(\vec{q}_{11} - \vec{q}_{12})\, t_1 + (\vec{q}_{22} - \vec{q}_{21})\, t_2 = \vec{q}_{11} - \vec{q}_{21}$$

soit:

$$\begin{pmatrix} x_{11} - x_{12} & x_{22} - x_{21} \\ y_{11} - y_{12} & y_{22} - y_{21} \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} x_{11} - x_{21} \\ y_{11} - y_{21} \end{pmatrix} \tag{6.51}$$

soit

$$D = \begin{vmatrix} x_{11} - x_{12} & x_{22} - x_{21} \\ y_{11} - y_{12} & y_{22} - y_{21} \end{vmatrix} \tag{6.52}$$

$$D_1 = \begin{vmatrix} x_{11} - x_{21} & x_{22} - x_{21} \\ y_{11} - y_{21} & y_{22} - y_{21} \end{vmatrix} \tag{6.53}$$

$$D_2 = \begin{vmatrix} x_{11} - x_{12} & x_{11} - x_{21} \\ y_{11} - y_{12} & y_{11} - y_{21} \end{vmatrix} \tag{6.54}$$

Si: $D \neq 0$: alors:

$$t_1 = \frac{D_1}{D} \tag{6.55}$$

$$t_2 = \frac{D_2}{D} \tag{6.56}$$

de l'équation 6.49 (ou 6.50), 6.55, 6.56 on aura: $\vec{s}_2(x, y)$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{11} \\ y_{11} \end{pmatrix} + t_1 \begin{pmatrix} x_{12} - x_{11} \\ y_{12} - y_{11} \end{pmatrix} \tag{6.57}$$

Si :

$$0 \leq D_1 \leq D \tag{6.58}$$

$$0 \leq D_1 \leq D \tag{6.59}$$

on aura:

$$0 \leq t_1 \leq 1 \tag{6.60}$$

$$0 \leq t_2 \leq 1 \tag{6.61}$$

Le **sommet** de convexité existe et ses coordonnées sont fournies par l'équation 6.57 et les relations 6.49 ou 6.50.

Si: $D = 0$, c'est à dire que les 2 segments $q_{11}q_{12}$ et $q_{21}q_{22}$ sont parallèles, et si on a:

$$D_1 \neq 0 \tag{6.62}$$

$$D_2 \neq 0 \tag{6.63}$$

c'est à dire que les 2 segments sont écartés, il n'y aura pas de solution. Sinon, les 2 segments $q_{11}q_{12}$ et $q_{21}q_{22}$ sont collés ensemble, on aura plusieurs solutions sur la partie commune aux deux segments $q_{11}q_{12}$ et $q_{21}q_{22}$. Le calcul du segment commun sera détaillé ci-dessous; dans ce cas, le domaine de convexité sera dans le triangle dont la base est le segment commun et le sommet, le sommet du carrefour.

segment commun du sommet de convexité

Dans le cas où le sommet de convexité se situe sur un segment, on veut le connaître, on reprend deux segments dans la section précédente, $q_{11}q_{12}$ et $q_{21}q_{22}$.

Si: $\exists t_1, t_2$, pour

$$0 \leq t_1 \leq 1 \tag{6.64}$$
$$0 \leq t_2 \leq 1 \tag{6.65}$$

tels que:

$$\vec{q}_{11} = \vec{q}_{21} + t_1(\vec{q}_{22} - \vec{q}_{21}) \tag{6.66}$$
$$\vec{q}_{12} = \vec{q}_{21} + t_2(\vec{q}_{22} - \vec{q}_{21}) \tag{6.67}$$

alors, le segment $q_{11}q_{12}$ est le segment commun.

D'autres cas existent.

Simplification de calcul de sommet de $\vec{s}_2$

Pour avoir une autre formule de l'équation 6.57, soit k_1 la pente de la base $q_{11}q_{12}$ du triangle(q_{11}, s_0, q_{12}), et k_2 celle de la base du triangle(q_{21}, s_0, q_{22}):

$$k_1 = \frac{y_{11} - y_{12}}{x_{11} - x_{12}} \tag{6.68}$$
$$k_2 = \frac{y_{21} - y_{22}}{x_{21} - x_{22}} \tag{6.69}$$

le point d'intersection de ces deux base:

$$y = k_1 x + y_{12} - k_1 x_{12} \tag{6.70}$$
$$y = k_2 x + y_{22} - k_2 x_{22} \tag{6.71}$$

donc, on peut avoir les coordonnées du sommet de convexité s_2:

si $k_1 \neq k_2 \neq 0$, alors :

$$y = \frac{y_{12}k_2 - y_{22}k_1 + k_1k_2(x_{22} - x_{12})}{k_2 - k_1} \tag{6.72}$$
$$x = \frac{y_{12} - y_{22} + x_{22}k_2 - x_{12}k_1}{k_2 - k_1} \tag{6.73}$$

6.6.3 Distance entre carrefours

Un carrefour est marqué par un sommet qui est le point de croisement de toutes les routes qui passent en ce point; il est plutôt un objet ponctuel que linéaire malgré ses composantes linéaires.

Naturellement, la distance entre deux carrefours est défini par la distance entre les deux sommets.

Mais, si le raisonnement s'effectue sur les axes de routes, ces routes sont représentées graphiquement par des signes conventionnels d'une certaine largeur; les carrefours auront donc une certaine surface.

Les signes conventionnels des routes interviennent largement dans la perceptibilité des carrefours et la séparation de deux carrefours voisins.

Comme les signes conventionnels amplifient de plus en plus leur emprise graphique au fur et à mesure que l'échelle diminue, les carrefours amplifient leur tailles et deux carrefours voisins se rapprochent de plus en plus jusqu'à leur confusion; ainsi se créent des ambiguïtés.

Pour que les formes des carrefours soient distinguées, et que deux voisins soient séparés, on doit élargir les distances entre deux carrefours au moins au seuil de séparations défini dans la section 2.4.

On définit d'abord la notion d'**axe de deux carrefours**.

Définition 15 *L'axe de deux carrefours est le segment dont les deux extremités sont les sommets de ces deux carrefours.*

Le segment p_2p_5 est l'axe de deux carrefours au point p_2 et au p_5 respectivement sur la figure 6.22 de la page 126.

Définition 16 *La distance entre deux carrefours est la plus courte distance entre les bords de routes de ces 2 carrefours, comptée parallèlement à l'axe de ces 2 carrefours.*

Compte tenu des effets des largeurs de signes conventionnels des routes, on définit la distance nette entre 2 carrefours comme la distance entre leurs sommets en déduisant les segments de décalages (défini au paragraphe 6.4.4) qui sont les effets de projections des largeurs des routes. (voir la figure 6.6 sur la page 104).

Cas de deux carrefours non liés à leur axe

Le segment de décalage à déduire est la plus grande projection de la largeur d'une des routes sur l'axe de ces 2 carrefour, cet axe était imaginé comme un arc sans largeur.

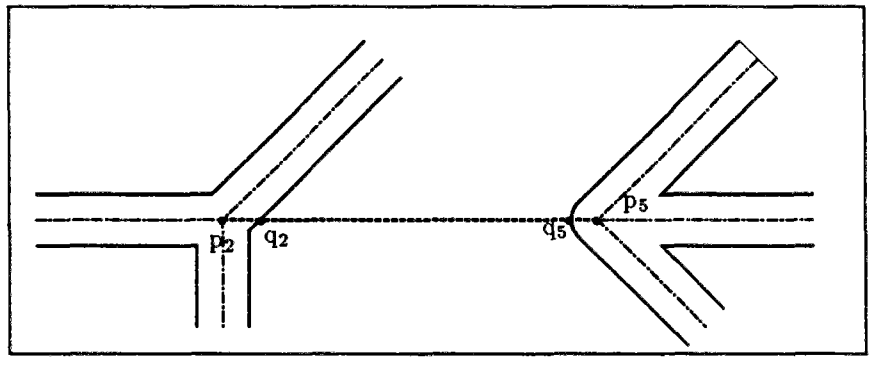


Figure 6.22: Distance de 2 carrefours non liés à leur axe

Propriété 5 *si ces deux carrefours ne sont pas reliés par un segment à leur axe, l'axe de projections des routes de deux carrefours se trouve sur l'axe de ces deux carrefours.*

La distance de ces deux carrefours est donc q_2q_5 sur la figure 6.22 de la page 126.

Cas de deux carrefours reliés à l'axe

Le segment de décalage à déduire est la plus grande projection d'une des routes avec sa largeur sur l'arc (un segment droit) qui relie les deux sommet; cet arc a une lageur, donc la projection d'une autre route sur cet arc est sur un des bords de l'arc qui relie ces 2 carrefours.

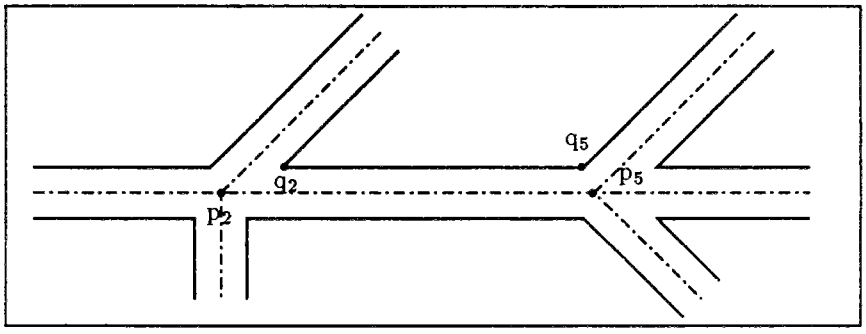


Figure 6.23: Distance de 2 carrefours reliés à l'axe

Propriété 6 *si ces deux carrefours sont reliés par un segment, la distance entre ces 2 cerrefours est la longueur de l'axe diminuée des 2 segments de déclages*[2].

La distance du segment q_2q_5 est la plus courte distance entre les deux carrefours au point p_2 et au p_2 respectivement sur la figure 6.23 de la page 127.

La distance de ces deux carrefours est donc q_2q_5.

6.6.4 Direction de déplacement d'un carrefour

Quand on bouge un point de croisement, ça va changer les formes de toutes les routes qui passent par ce croisement, par conséquent, les convexités des routes vont être changées, pour garder leur convexité, on a une définition ou plutôt une propriété suivante:

Définition 17 *Pour garder la convexité d'un carrefour, la meilleur direction de déplacement d'un carrefour se situe dans son* **domaine de convexité** *s'il existe.*

Quand on déplace un carrefour, souvent c'est pour l'écarter d'un autre objet, une route, un autre point de croisement, etc. La direction d'écartement ne tombe pas forcément dans son **domaine de convexité**. Il faudra donc trouver un compromis ou envisager une translation de l'ensemble des routes de ce carrefour. Les directions de déplacement d'un carrefour doivent toujours garder les convexités de chaque route qui arrive sur ce sommet de carrefour, elles sont choisies en respectant les règles globales 1, 2, 3 définies dans la section 6.4.3.

[2]défini au paragraphe 6.4.4

6.7 Stratégie générale

La stratégie générale est que tant qu'il y a des conflits entre les différents objets, des règles de résolutions seront déclenchées.

La figure 6.24 montre la conséquence de cette stratégie. La chapitre 9 montre des exemples réels à titre d'illustration.

6.8 Résumé

En conclusion, l'harmonisation réside dans la conservation de la solidarité de tous les éléments de la carte, dans l'équilibre entre ces différents éléments tout en maintenant le caractère géographique local et en évitant l'égalisation et l'uniformisation. Si la sélection et l'harmonisation ont été bien menées, ces conditions seront plus faciles à respecter.

La généralisation intervient au stade de la conception de la carte (définition des critères de sélection) et pendant son élaboration et sa rédaction, d'une façon très complexe. Les problèmes d'avenir de la cartographie automatique ne seront résolus, en donnant satisfaction aux auteurs, aux réalisateurs et aux utilisateurs, que dans la mesure où des règles et des principes auront codifié la généralisation sous ses multiples aspects.

L'établissement de cette méthodologies sera une des tâches essentielles du cartographe et aussi des informaticiens de demain.

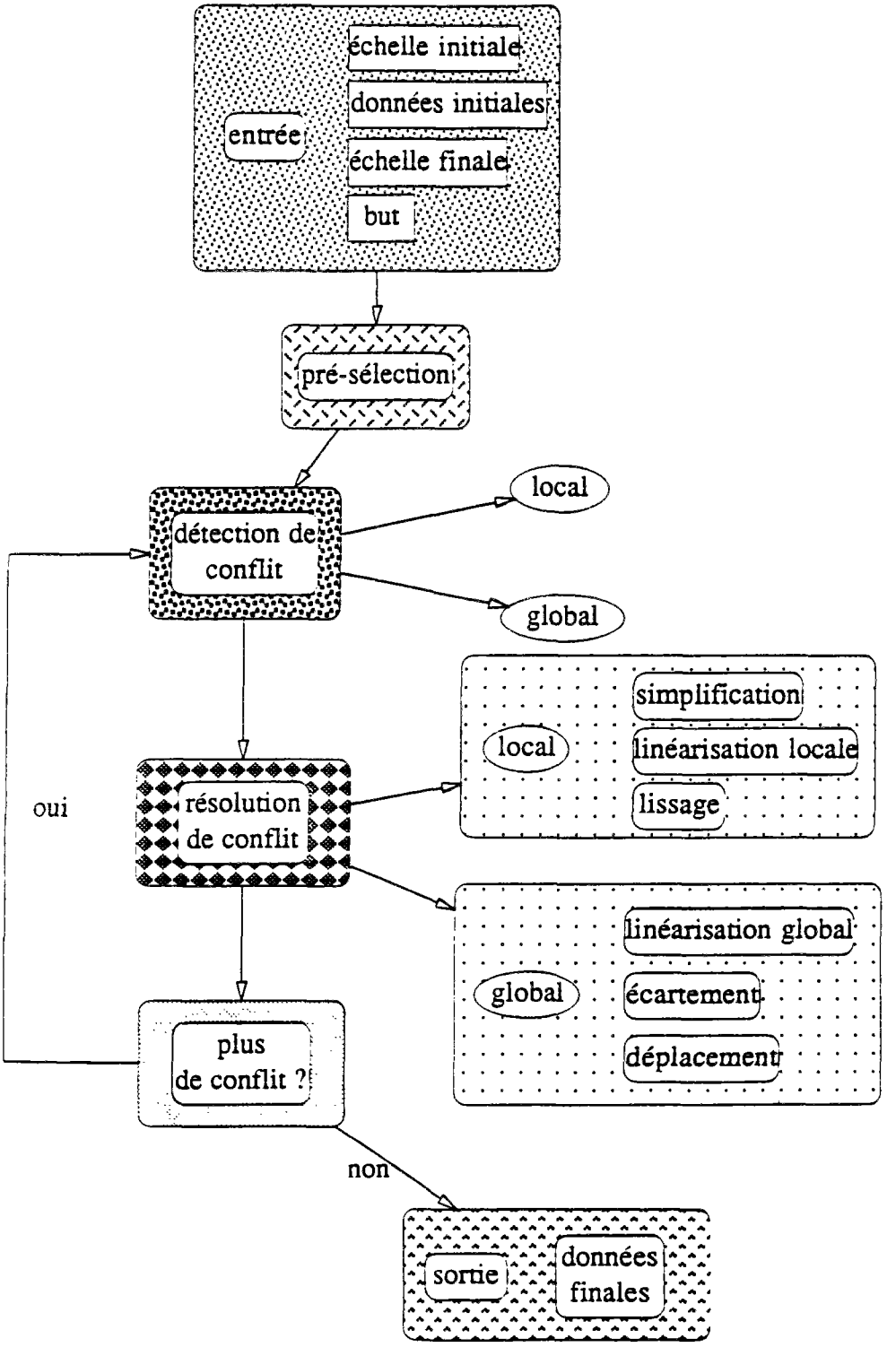


Figure 6.24: Stratégie de résolution

Chapitre 7

La base de connaissances

7.1 Les structures des objets

Nous avons introduit la notion d'objet dans la section 3.3. Le concept d'objet repose sur le principe d'encapsulation: l'objet regroupe au sein d'une entité unique des données et les procédures qui les manipulent. Il communique avec le monde extérieur à travers une interface, qui cache son organisation interne, assurant ainsi l'abstraction des données.

En ce qui concerne les objets cartographiques, le principe est qu'un objet cartographique ne doit pas être un simple objet graphique, il doit être doté des connaissances les plus complètes possibles: statistiques, graphiques et géographiques. Un objet géographique n'est jamais seul, il est toujours en relation étroite avec les autres objets de la carte.

Un cartographe, avec l'ensemble des éléments de la carte sous ses yeux, effectue par un processus mental unique plusieurs tâches presque parallèlement sur plusieurs éléments. Dans ce processus, les structures de l'information sont implicitement reconnues par le cartographe avec ses diverses qualités physiques: des yeux pour voir, un cerveau pour réfléchir, des mains pour dessiner. Il applique ensuite un ensemble des règles pratiques en fonction de l'ensemble de ces éléments. Ces règles, reposant sur l'expérience et sur la tradition, sont en général comme interface entre des séries de cartes classiques. Elles sont par conséquent étroitement liées à la forme des éléments et à leur structuration.

Pour bien modéliser les règles objectives du processus mental de la généralisation conduisant à une généralisation logique, la reconnaissance de structure est un élément important: une modélisation des objets auquels les règles vont s'appliquer est d'abord indispensable. La reconnaissance de structures complexes ne peut se faire sans une structuration des données appropriées. Un objet doit donc avoir les informations telles que sa forme individuelle géométrique, ses caractéristiques statistiques personnalisées, ses instructions graphiques, sa relation de proximité avec son voisinage, son importance individuelle et relative, et éventuellement la structuration de l'ensemble des éléments de la carte.

Il doit *connaître* ses voisins, deux cas étant possibles:

- il ne connaît peut être pas exactement les caractéristiques de ses voisins, mais

il doit savoir au moins leur nom, par la possibilité de structure d'un objet, (par exemple, la caractéristique d'*auto-typage* d'un objet **Ceyx**), on connaîtra la catégorie de ses voisins, et les caractéristiques normales communes à cette catégorie seront ensuite obtenues par des requêtes évoquées par des processus prédéfinis.

- un objet ne connaît pas du tout ses voisins, ni leur nom, ni leur existence. Un processus spécial de détection est donc nécessaire, il doit au moins fournir le nom d'un voisin pour qu'un autre processus de requête puisse fonctionner pour pouvoir fournir des informations.

La structure d'un objet et celle de la base des objets déterminent donc directement l'efficacité et la capacité d'un système pour la généralisation qui se sert de ces objets.

On décrit dans la section suivante les différents objets élémentaires qui seront les éléments de la base de connaissances, ces éléments permettant de modéliser les informations sous des formes plus naturelles et plus aisées à manipuler.

7.1.1 Organisation des objets

Les entrées et sorties du système d'une part et le raisonnement modélisé d'autre part, définissent plusieurs zones d'expertises pour le problème du réseau de communication:

- une zone des objets géométriques élémentaires neutres.

atomique : **Point**

composite : **Segment, Arc, Axe**, ils sont composés des objets atomiques: **Point**.

Ils n'ont que des coordonnées métriques, leur définitions se trouvent dans la section 6.4.1.

- une zone des objets géographiques: **objet du Réseau de communication**. Ce sont des objets composites ayant un sens géographique, et ils sont composés des éléments de la zone des objets géométriques. Cette zone est divisée en 3 zones:
  - zone **ferrée**
  - zone **routière**
  - zone **hydraulique**

Ces zones d'expertises sont définies par des *classes* ou des *sous-classes* au sens du langage orienté objet. Il existe sur le marché de nombreux langages orientés objets,

Les objets de différentes zones sont organisés hiérarchiquement. La *relation d'héritage* lie une classe à sa superclasse. La représentation graphique de la relation forme le *graphe d'héritage*, dont la figure 7.1 de la page 134 montre un exemple pour décrire les objets du réseau de communication, le graphe d'héritage est un arbre, puisque l'héritage est simple, étant données les contraintes de logiciel (Ceyx), chaque classe ayant une mère. Sa racine représente la classe la plus générale, appelé **Objet du réseau**. Cette classe détient le comportement commun à tous les objets d'un réseau. Elle spécifie n'importe quel objet du réseau de communication, il y a deux variables

d'instance **nom** et **éliminable?**, donc par l'héritage statique des variables d'instance, tout objet du réseau de communication aura un nom désigné par la variable d'instance **nom** et aura une propriété si cet objet est **éliminable**.

Les différentes classes s'organisent entre elles sous la forme d'un réseau hiérarchique construit sur une relation de filiation pour profiter des avantages de l'héritage des propriétés d'objets prédécesseurs et donc simplifier la représentation de chaque objet pris séparément. La relation d'héritage est *transitive*: les caractéristiques des classes supérieures sont héritées par les classes inférieures, qui sont d'autant plus spécialisées qu'elles sont proches des feuilles de l'arbre.

Par exemple: les superclasses de la classe **Autoroute** sont ordonnées de la façon suivante:

Autoroute → Route-à-2-chaussées → Route → Voies-routières

Voies-routières → Voies-de-communication → Objet-communication

Objet-communication → Objet-du-réseau.

Cette classe possède 11 variables:

- **payant?** qui lui est propre, cette variable peut avoir la valeur **oui** ou **non**.
- **type-chaussée**: héritée de la classe **Route-à-2chaussées**, cette variable prend une des valeurs suivantes:

  1 : inconnu

  2 : chaussées éloignées

  3 : chaussées séparées

  4 : autre

- **circulation**: héritée de la classe **Route**: son domaine de valeur est: **grande, moyenne, petite**.
- **code-ad**: héritée de la classe **Voies routières**, son domaine de valeur est:

  1 : inconnu

  2 : Nationale

  3 : Départementale

  4 : Communale

  5 : Urbaine

- celles héritées de la classe **Voies de communication**:

largeur : la largeur totale de la voie.

axe : la liste des points (ou segments) qui composent l'axe médian, les points sont des objets de la classe **Point**.

arcs : la liste des arcs[1].

[1] défini au paragraphe 6.4.1

noeuds : la liste des points sensibles qui sont les objets des sous-classes de la classe **Noeud de communication**: par exemples: les carrefours à niveau, les échangeurs, etc.

voisins-l : la liste des voisins linéaires tels que voies de communications.

voisins-p : la liste des voisins ponctuels.

- **nom** et **éliminable** héritées de la classe **Objet de communication**.

Par une extension de **Ceyx**, chaque classe peut posséder, outre les variables d'instance, des variables de classe[2] au sens **Smalltalk** ou champs "static" de **C++**. Ces variables, avec leurs valeurs, sont partagées par l'ensemble des instances de la classe et celles de ses sous-classes. Cela évite de donner une valeur à une variable chaque fois qu'on crée une nouvelle instance. Comme les variables d'instance, les variables sont héritées statiquement par les sous-classes, leurs valeurs peuvent être redéfinis au niveau des sous-classes sans changer celles de leur sur-classes.

Par exemple la classe **Voie de communication** possède une variable de classe **signe conventionnel**, donc toutes ses sous-classes possèdent cette variable de classe, et toutes les instances d'une classe partagent la même valeur héritée d'une sur-classe ou redéfinie au niveau de cette classe. A l'échelle de 1:250 000, si on donne à la variable de classe: **signe conventionnel** la valeur (20 30 12 30 20) à la classe **Autoroute**, et la valeur (8 40 8 40 8) à deux classes: **Voie expresse** et **Périphérique**, les instances des trois classes en question seront donc représentées différemment. Alors qu'à l'échelle de 1:1 000 000, si on donne à la même variable de classe la valeur (20 30 20) (deux traits de 20 mm séparés par un espace de 30 mm) à la classe **Route à chaussées séparées**, toutes les autoroutes, les voies expresses, les périphériques urbaines qui sont des sous classes de la classe **Route à chaussées séparées** seront donc indifféremment représentées par deux traits de 20 mm séparés par un espace de 30 mm.

Par la suite on distingue les variables d'instance des variables de classe.

Pour la recherche d'une propriété d'un objet, l'ordre du parcours de la hiérarchie est bien fixé, les classes sont bien évidemment examinées dans l'ordre établi, de la plus spécifique à la plus générale, en s'arrêtant à la première qui possède la propriété cherchée: la première occurence de la propriété cherchée masque les autres.

Concrètement, la hiérarchie d'héritage est parcourue de bas en haut, à partir de la classe concernée, en suivant les liens d'héritage.

Les valeurs de certaines propriétés (variables d'instance ou éventuellement variable de classe) peuvent être des pointeurs vers d'autres objets prédécesseurs ou successeurs, ou encore une indication des exceptions abordées dans la section 4.11. Par exemple, on définit une variable d'instance **maitre-route** pour la classe **Arc**, cette variable pointe donc vers la route qui contient cet arc.

Quelques descriptions non complètes de définitions de classe sont illustrées ci-dessous:

La classe **Reseau de communication**

```
?(deftclass Reseau-C nom~string)
= #:Tclass:Reseau-C
```

Ce qu'on sait sur cette classe:

[2] voir la section 5.2 où est décrite une implantation de cette variable

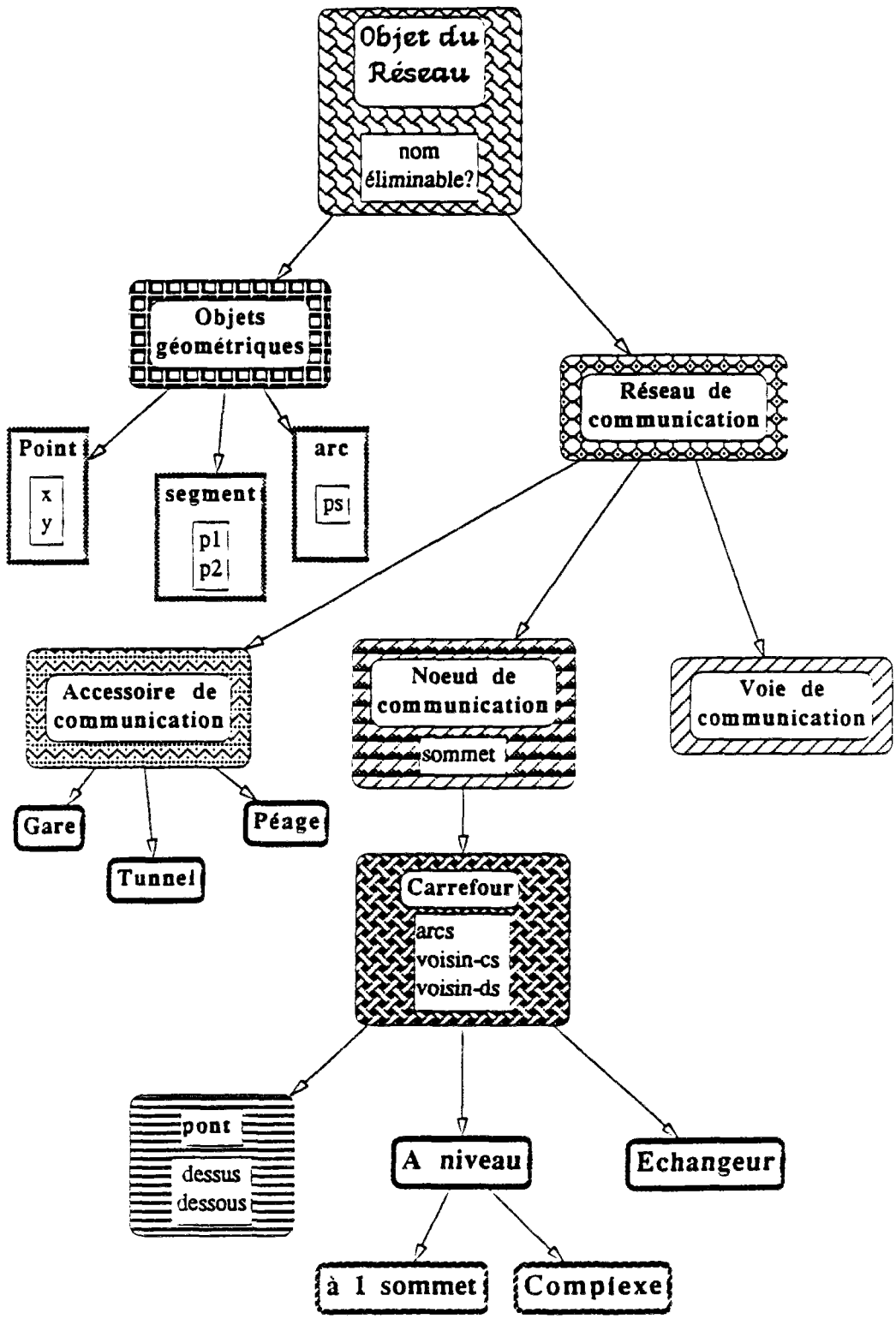


Figure 7.1: Les objets du réseau de communication

```
          Type: #:Tclass:Reseau-C

Abreviations:
     Reseau-C

Champs:
     class-attributes ~ (Vector ...)
     nom ~ string

Proprietes Semantiques:
     nom obj

Sous Modeles:
     Noeud-C
     Voie-C
```

La classe Noeud de communication

```
? (deftclass {Reseau-C}:Noeud-C
     centre~symbol larcs~cons voisin-cs~cons voisin-ds~cons)
= #:Tclass:Reseau-C:Noeud-C
```

Ce qu'on sait sur cette classe:

```
          Type: #:Tclass:Reseau-C:Noeud-C

Abreviations:
     Noeud-C

Champs:
     class-attributes ~ (Vector ...)
     nom ~ string
     centre ~ symbol
     larcs ~ cons
     voisin-cs ~ cons
     voisin-ds ~ cons

Proprietes Semantiques:
     centre obj
     larcs obj
     voisin-cs obj
     voisin-ds obj

Sous Modeles:
     Carrefour
     Gare
     Peage
     Ville
```

La classe Voie de communication:
La figure 7.3 de la page 137 montre les classes et sous-classes de cette sur-classe.

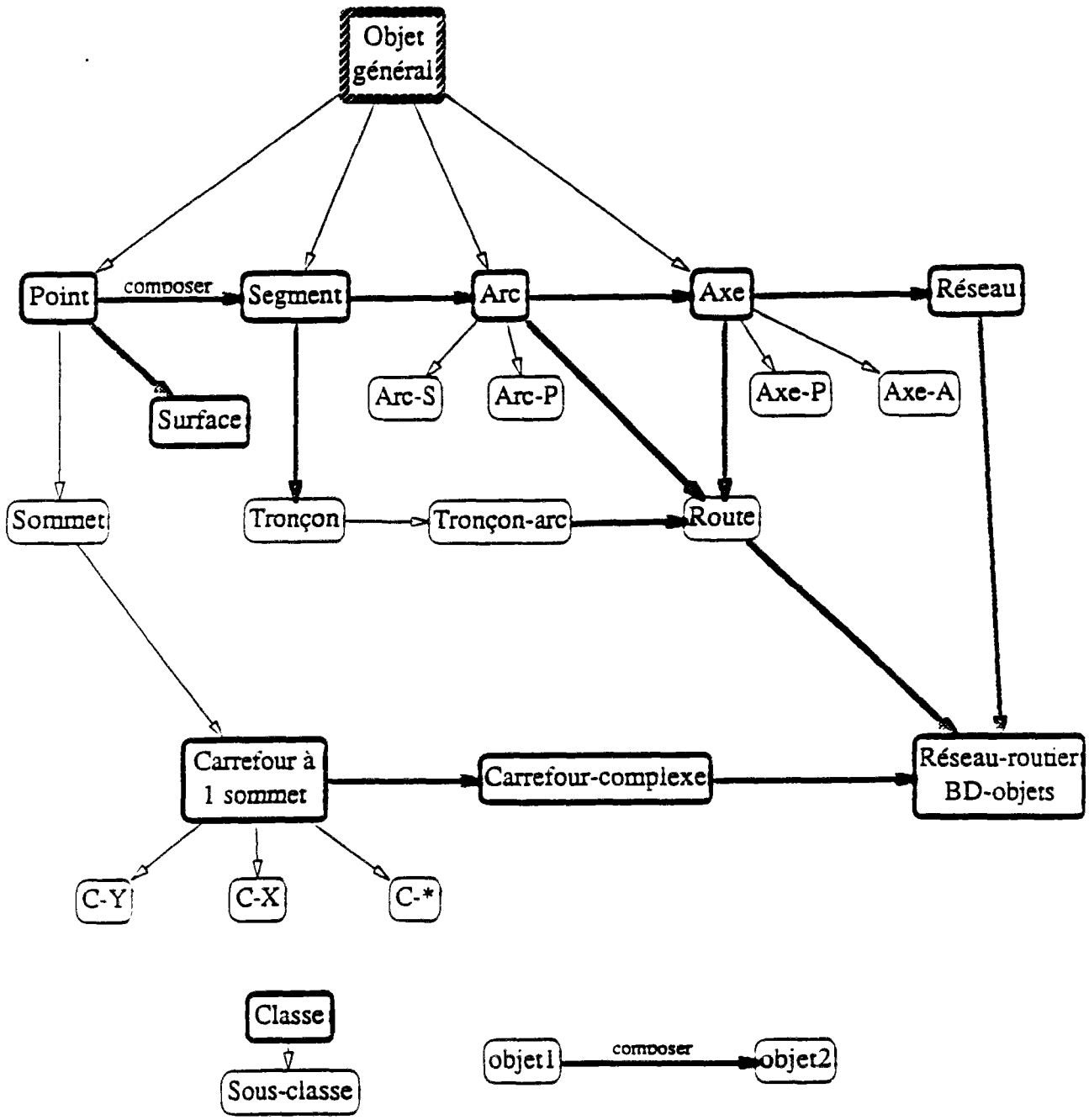


Figure 7.2: Les compositions des objets du réseau

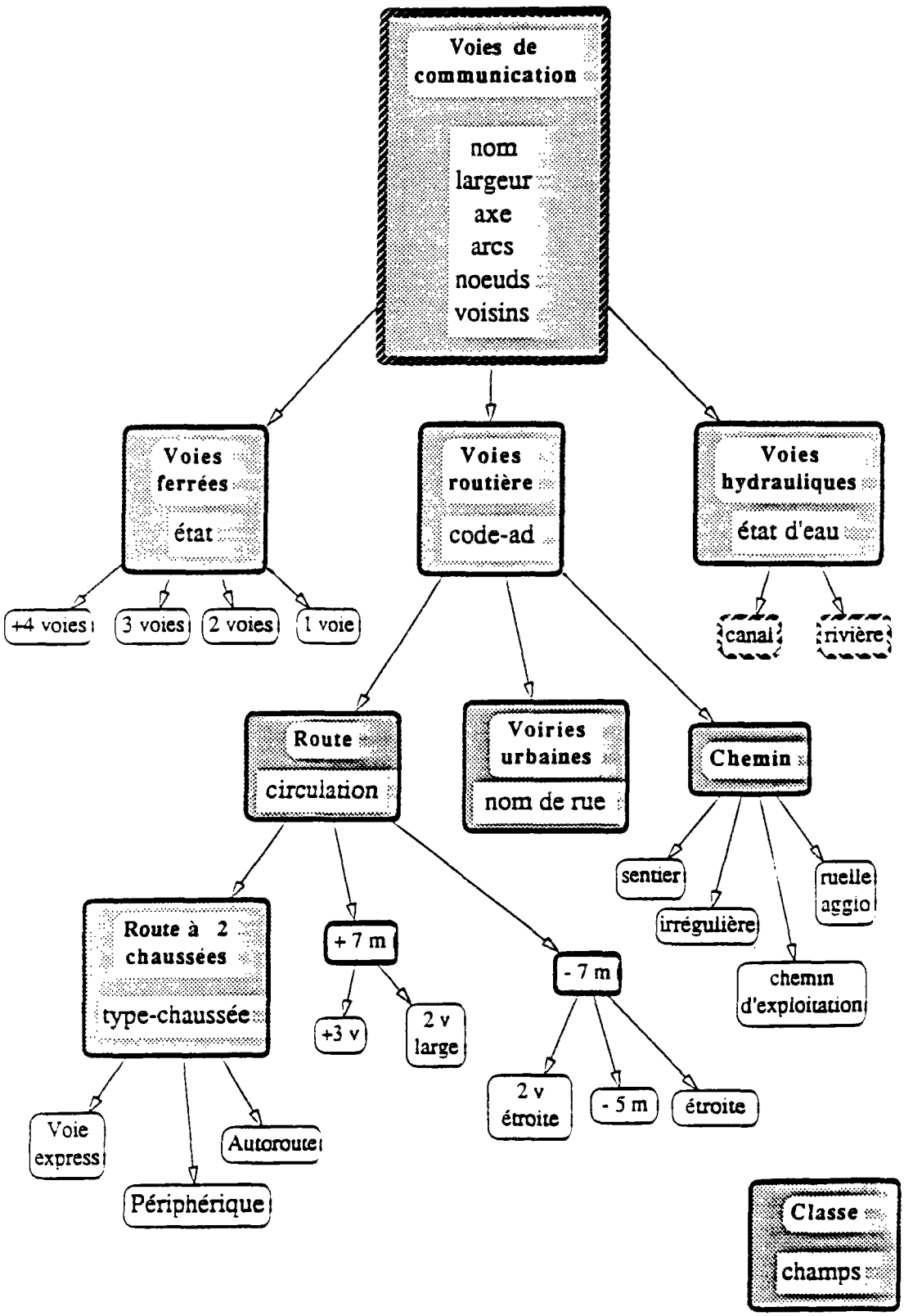


Figure 7.3: Les classes et sous-classes de `Voies de communication`

```
(deftclass {Reseau-C}:Voie-C
      largeur~fix lvoisins~cons axe~cons larcs~cons lncs~cons)
 = #:Tclass:Reseau-C:Voie-C
```

Ce qu'on sait sur cette classe:

```
      Type: #:Tclass:Reseau-C:Voie-C

Abreviations:
     Voie-C

Champs:
     class-attributes ~ (Vector ...)
     nom ~ string
     largeur ~ fix
     lvoisins ~ cons
     axe ~ cons
     larcs ~ cons
     lncs ~ cons

Proprietes Semantiques:
     axe obj
     larcs obj
     largeur obj
     lncs obj
     lvoisins obj

Sous Modeles:
     Voie-ferree
     Voie-hydraulique
     Voie-routiere
```

7.1.2 Les objets composites

Composer des objets est particulièrement courant en représentation des connaissances, mais assez souvent, les facilités font cruellement défaut dans les langages.

L'héritage multiple permet de définir de nouveaux objets par fusion de la structure et du comportement de plusieurs autres objets. Mais l'héritage permet seulement de définir des classes d'objets qui sont presque semblables à d'autres. Les liens d'héritage représentent, grosso-modo, des inclusions entre ensembles, une spécialisation entre types. Mais on ne peut pas aller plus loin, l'héritage est inapproprié à la représentation de liens entre un objet et ses composants. Par exemple: une **Ville** est composée d'une mairie, des réseaux routier, des parcs, des réseaux ferrés, des services publics. Cependant c'est une erreur de penser que le comportement d'une ville soit l'union des comportements de ses différents éléments. La classe **Ville** ne peut pas être définie comme la spécialisation de la classe **Réseau routier**, ni de la classe **Autoroute**, ni de l'une des autres classes, car une ville n'est pas une route. En revanche, un réseau routier appartient partiellement à une ville, une **Autoroute** est bien une **Route à chaussées séparées**, une **Route** générale, une **Voie routière**, et une **Voie de communication**,

donc une **autoroute** a un comportement qui est l'union des comportements de ses sur-classes évoquées ci-dessus.

On dispose quand même déjà d'un moyen pour représenter un objet composite: les variables d'instance. Une classe décrivant structurellement une ville est pourvue de variables pour désigner ses différents comportements. Les différents éléments d'une ville seront représentés par des variables d'instances correspondantes: **mairie**, **réseau**, **routier**, **réseau ferré**, **réseau hydraulique**, etc. Une ville peut donc avoir autant de route qu'elle veut et qu'elle peut posséder(?).

7.2 Les procédures des objets

Chaque classe d'objets a une liste de procédures (méthodes):

- soit pour fournir de simples paramétres concernant un objet. Par exemple, livrer la valeur de la largeur d'une route.
- soit pour répondre si nécessaire à des requêtes ponctuelles sur cet objet telles que: une route a-t-elle un ou des voisins linéaires à proximité? Un déclenchement d'autres méthodes sera éventuellement possible.
- soit des actions du type opérations de généralisation telles qu'une sélection des objets à conserver, ou des algorithmes candidates bien définis qui sont des opérations graphiques (des modélisations des opérations dans le chapitre 6) tels qu'un lissage local d'une ligne.
- soit des méthodes de choix des opérateurs ou des algorithmes graphiques de généralisation. Par exemple, quand il y a une superposition entre deux routes, on veut savoir si on peut résoudre ce problème par une translation d'une des 2 routes dont l'importance est moins grande, ou si une suppression d'une route d'importance inférieure est à envisager.

On distingue des objets géométriques et des objets géographiques, par conséquent, on distingue des méthodes statistiques et des méthodes cartographiques. Les objets cartographiques sont des objets composites construits à partir d'objets géométriques, et les méthodes cartographiques sont des méthodes basées sur les méthodes statistiques. C'est-à-dire que les méthodes cartographiques peuvent appeler des méthodes statistiques.

Par exemple, un objet **arc** de la classe **Objet géométrique** a ses méthodes telles que: linéarisation, déplacement, etc; pour un objet composite **arc d'autoroute** doté d'un **arc** et d'autres variables, cet objet **arc d'autoroute** aura donc la possibilité de faire une requête ou demander des renseignements sur l'arc géométrique associé.

Les opérations de généralisation sont évoquées de deux façons: explicitement ou déclarativement.

Les procédures opératoires, bien définiees telles que *Déplacer* une route sont désignées et appelées explicitement par leur nom sous forme de messages qu'on envoie en cas de nécessité à l'objet: route.

Si on sait pertinemment que telle connaissance, précisément déterminée et assurément présente dans le système, doit être invoquée sans possibilité d'alternative, la rédaction déclarative n'aura pas la nécessité d'avoir lieu. Ces procédures des objets

peuvent être appelées dans les règles sous formes toujours de messages aux objets correspondants. Par contre les décisions de déclencher ces procédures ponctuelles sont évoquées par des règles, de façon déclarative. C'est-à-dire que le séquencement des opérations de généralisation est contrôlé par des règles de décision, appelant des algorithmes applicables en fonction des situations des éléments cartographiques concernés et de précisions demandées.

7.3 Exemples: les sommets-carrefours

Les modèles sont décrits à l'aide d'une liste de champs et d'une liste de méthodes (procédures); par exemple de l'extérieur, la classe de Carrefour-élémentaire (à un sommet) aura des:

champs :

- nom,
- sommet,
- liste-d'arcs,
- voisins-cercles,
- voisins-reliés,
- directions-de-déplacement-autorisées,
- fréquentation,
- etc.

procédures :

1. *Décrire-toi*: qui livre la structure de ce carrefour.
2. *Sortie.ps*: qui donne les codes en PostScript pour son toponyme.
3. *new-voisin-cercles*: qui cherche s'il a de nouveaux voisins de cercle.
4. *new-voisin-directs*: qui met à jour les voisins directement reliés en cas d'opérations qui risquent de changer son environnement.
5. *S1>S2?*: qui décrit la priorité relative par rapport à un Sommet-Carrefour donné.
6. *HorsRond*: qui donne la liste des arcs de Sommet-C1 qui sont en dehors du Rond centré à un autre Sommet-C2, avec la distance de deux Sommet-Cs comme diamètre; donc ces arcs sont disponibles pour écarter ce Sommet-C1 en éloignant du Sommet-C2.
7. Visualise-toi: qui visualise le carrefour graphiquement sur un terminal.
8. *Type-nb*: qui donne le nombre exact de chaque type d'arc à un moment donné. Par exemple: ((A . 2) (N . 0) (R . 0) (D . 4)) signifie qu'il y a 2 arcs de type Autoroute, 4 arcs de type Départementale.
9. Déplacer-toi: qui déplace le carrefour dans un endroit donné.
10. Déplacer-unarc: qui déplace seulement un arc spécifique.
11. Ecarter-toi: déplace en s'éloignant d'un autre objet.

Sous forme interne, on définit les propriétés sémantiques d'un modèle comme étant l'ensemble des fonctions définies dans l'espace de noms du modèle, ainsi définir une propriété sémantique du modèle revient à définir une fonction Lisp classique dans l'espace de noms du modèle.

En Le-lisp, les espaces de noms sont organisés de manière arborescente à l'aide de packages, ce qui se traduit par une organisation hiérarchique des espaces sémantiques associés aux modèles. Les propriétés sémantiques d'un modèle constituent en quelque sorte son mode d'emploi: comment accéder aux champs des instances, comment imprimer ces instances, comment les éditer, etc.

Il est de bon ton de considérer qu'une propriété sémantique soit étiquettée par le nom du modèle (ou de la classe) sur lequel on définit cette propriété sémantique: ainsi, on pourra envoyer des messages pour déclencher des procédures relatives aux objets typés.

Pour appliquer une fonction de manipulation à une instance, on lui envoie en message le nom de la fonction et ses arguments éventuels; l'objet, qui connaît son type, n'a donc plus qu'à chercher dans l'espace sémantique associé à ce type la fonction voulue.

Un des traits essentiels des objets est qu'ils possèdent implicitement toutes les informations et comportements nécessaires. De plus ils composent en réseau de modèle; si la recherche d'une valeur échoue, des procédures de remplacement, les mecanismes d'héritages des objets ou d'actions des règles, peuvent être utilisées.

7.4 Formuler les connaissances

7.4.1 Formuler les faits

Sous le terme commun de *fait*, on désigne des expressions et des interprétations des expressions:

- des connaissances assertionnelles,
- utilisables pour conditionner puis réaliser l'exploitation de connaissances opératoires,
- pouvant être créés et détruits par l'exploitation de connaissances opératoires.

Pour simplifier, on utilise des prédicats[3] pour représenter des faits au lieu de langage naturel. Un fait est donc composé d'un prédicat et des arguments. Par exemple, le fait sous forme de langage naturel

la route-x et la route-y se sont superposées

sera présenté par :

```
Superposer(?route-x, ?route-y)
```

avec **Superposer** comme prédicat, et route-x, et route-y comme ses deux arguments, le point d'interrogation ? désigne une variable. Les valeurs des arguments seront des instances des classes prédéfinies telle que **Autoroute**, **sentier**, etc.

La section 7.4.2 détaille la définition et l'implantation des **prédicats**.

[3] voir la section 7.4.2

7.4.2 Les prédicats

La perception globale de la figure (carte) par un cartographe lui suggère les ordres des éléments de la carte et les opérations nécessaires de la généralisation en appliquant des règles implicitement. La résolution du problème par une machine (un système expert par exemple) ne consiste pas à balayer aléatoirement toute la base de règles, du moins nous ne pensons pas que l'opérateur humain procède ainsi. La considération des données du problème nous indiquera quel groupe de règles utiliser pour résoudre le problème en question. On se sert ici des **prédicats** pour déterminer ces règles. C'est en particulier l'ensemble des prémisses communes qui se factorise dans une règle; en recommençant la factorisation sur les prémisses restantes, on obtient une cascade de contextes qui forme ainsi le réseau expert.

Les prédicats sont utilisés pour représenter les faits de la base de connaissances et les opérateurs du processus de la généralisation tels que: écarter, agrandir, etc. en prenant éventuellement des arguments. Par exemple: (Ecarter s1 s2) représente le fait: déplacer le sommet s1 en s'éloignant de s2. Les règles sont groupées par les prédicats qui apparaissent dans la partie condition ou conclusion:

- Pour que les évaluations des états de système (état de règles, états de faits connus) soient efficaces, on cherche des améliorations dès le niveau de l'organisation des données, non seulement au niveau de l'implantation des faits et des objets élémentaires, mais aussi au niveau de la structuration des règles et de leurs composants: les prédicats. Les prédicats apparaissent dans les deux parties (prémisse ou conclusion) d'une règle mémorisent également l'état de cette règle.
- Pour limiter le temps souvent coûteux de filtrage, on a besoin de primitives permettant d'accéder plus vite à la base des règles, pour modifier aussi les modules de la base des règles valides, ainsi que toute partie de la base des connaissances. On se sert également des prédicats qui apparaissent dans les règles pour mémoriser l'état de la base des règles.

Les prédicats appartiennent à une classe de prédicats. Donc chaque prédicat est repéré par un symbole Lisp, les informations relatives à ce prédicat sont accrochées aux champs (variables d'instance ou variables de classe) ci-dessous:

nom: le nom est un symbole.

Lfaits-totale: contient l'ensemble de toutes les instances des faits qui contient ce prédicat.

Lreg-ante: liste des doublets dont les premiers termes sont les règles dans lesquelles le prédicat apparaît en partie antécédents tandis que les deuxièmes sont les nombres d'occurences de ce prédicat.

Lreg-conc: liste des doublets dont les premiers termes sont les règles dans lesquelles le prédicat apparaît en partie conclusions, et les deuxièmes, les nombres d'ocurrences de ce prédicat.

Par exemple: soit

la base de faits: (Ecarter s1 s2) (Ecarter s1 s3).

la base de règles: R3.1[4]

[4] sur la page 144

où s1, s2, s3 sont des objet de la classe de **Sommet-carrefour**. Dans le filtre: (Ecarter ?arg1 ?arg2) ou le prédicat **Ecarter** prend 2 arguments qui sont des variables simples, il aura la forme interne:

nom : Ecarter

Lfaits-totale : ((Ecarter s1 s2) (Ecarter s1 s3))

Lreg-ante : ((R3.1 . 2)) la règle R3.1 a deux occurences de ce prédicat en prémisses.

Lreg-conc : ()

Remarques:

1. Les prédicats variables comme dans (?P ?arg1 ?arg2) ne sont pas comptés dans le champ **Lreg-ante** ou **Lreg-conc**.
2. Le même prédicat qui apparaît plusieurs fois dans une même règle est compté dans les champs: **Lreg-ante** et **Lreg-conc** autant de fois qu'il apparaît dans cette règle.
3. Pour l'instanciation de prédicat variable, une nouvelle instance de cette variable ne donne pas une nouvelle chance pour déclencher la règle qui contient ce prédicat.

7.4.3 Formuler les règles

La figure 7.2 de la page 136 donne une description des règles de généralisation.

Le principal dénominateur pour formuler les règles réside dans ce que l'on peut appeler la rédaction déclarative des règles. C'est à dire que la désignation d'une information ne se fait pas par étiquette ou pointeur (c'est à dire un nom individuel externe) mais par la donnée d'un fragment de l'information:

- Chaque règle fournie par l'expert est rédigée de telle sorte que la représentation définit:
  - non seulement les conclusions (les effets) de la règle,
  - mais aussi les conditions de déclenchement, les conditions dans lesquelles cette règle est applicable.

  En usage normal, pour sélectionner une règle particulière, on ne fournit pas un nom individuel propre à cette règle mais un groupe de faits susceptibles d'être compatibles avec les conditions de déclenchement de la règle;
- Dans une règle on ne désigne jamais une autre règle par un nom individuel.

L'expression explicite des conditions de déclenchement dans la représentation de la règle correspond très souvent à la formulation naturelle par les experts de leur savoir-faire. Elle tend donc à favoriser la saisie des connaissances.

Simultanément elle favorise la révision des connaissances. En effet, il n'est pas nécessaire de donner un nom à chaque règle pour pouvoir la désigner lorsqu'on en a besoin. La rédaction déclarative des règles dans les systèmes experts permet, en

principe, de rédiger chaque règle (unité de savoir-faire) en ignorant l'existence effective des autres. On présume seulement que chaque règle introduite dans la base de connaissances comporte, quelque soit son auteur, des expressions de conditions qui permettront d'évaluer si la règle est applicable et donc, éventuellement, de l'appeler. On présume aussi que les faits introduits comme effets d'une règle peuvent contribuer à appeler d'autres règles. Le grand avantage de la rédaction déclarative des règles est de permettre d'ajouter et de supprimer aisément des règles sans être obligé d'étudier les conséquences des ajouts et de suppressions; elle confère aux systèmes experts une meilleure modularité.

Les connaissances dynamiques sont donc codées sous la forme de règles de production et de fonctions Lisp. Généralement, une règle en forme externe comporte deux parties:

Si *antécédents* **Alors** *conséquents*

On a abandoné la forme CONCLUSION-SINON par souci de simplification. La partie *antécédents* est une conjonction de conditions, on peut exprimer des disjonctions soit à l'intérieur d'une condition, soit en définissant plusieurs règles ayant la même partie conclusions. La partie *conséquents* peut être une conjonction de sous-conclusions ou un ensemble d'actions à envisager.

Grammaire des règles:

<déclaration-règle> ::= **règle**<identificateur> <règle>
<règle> ::= **Si**<antécédents> **Alors** <conséquents>
<antécédents> ::= [<prémisse> || <action >] [et <antécédents>]*
<prémisse> ::= < filtre > || non || echec ||
enrichir (<variable> < expr **Lisp** >)
<filtre> ::= (< prédicat > < élément > *)
<prédicat> ::= < identificateur> || ?<identificateur>
<élément> ::= <identificateur> || <variable> || <filtre> ||
/ <expr Lisp>
<action> ::= [execute || avec || / || prédicat-execution] (expr **Lisp**)*
<conséquents> ::= [<conclusion> || <action>] [et <conséquents>]*
<conclusion> ::= ajoute || supprime || conclut || (<filtre>)
<variable> ::= [? || &] <identificateur> || ?- || & -
<identificateur> ::= symbole qui ne débute pas par ? ou &

Exemple: la règle R3.1 groupe des actions :

R3.1 : *deux objets différents demandent de déplacer un même objet, alors un déplacement commun en éloignant de ces deux objets est proposé. (Ecarter ?obj1 ?obj2) veut dire qu'on doit écarter ?obj1 de ?obj2*

Si: (Ecarter ?obj1 ?obj2)
; *?obj2 demande ?obj1 de l'éloigner*
et (Ecarter ?obj1 ?obj3)
; *?obj3 le demande aussi*
et (nequal ?obj2 ?obj3)
; *?obj2 ?obj3 différents*
Alors:

Ajoute (Ecarter-g ?obj1 (?obj2 ?obj3))
;*on les groupe en évitant de déplacer 2 fois ?obj1*
Supprime (Ecarter ?obj1 ?obj2)
Supprime (Ecarter ?obj1 ?obj3)

Langages pour déclencheur

Ils jouent un rôle essentiel pour permettre la rédaction associatives des règles et donc faciliter un style de *programmation déclarative*: les règles sont indépendantes les unes des autres. On respecte un modèle de compatibilité entre les faits et les éléments du déclencheur d'une règle, on appelle les éléments *filtre*.

Dans notre système dit avec *variables*, seules les règles admettent des variables, mais pas les faits, et donc comme dans beaucoup de systèmes, on respecte le modèle de *semi-unification* entre les faits et les filtres, un cas particulier de l'*unification* utilisée en programmation logique, où les variables n'apparaissent que dans un des deux termes à unifier.

Les éléments qui composent les déclencheurs de règles, appelés filtres, respectent la compatibilité entre les données et les modèles. Les règles peuvent comporter explicitement des variables, ce qui facilite beaucoup l'expression des connaissances et pose donc des problèmes d'unification des expressions et des problèmes de filtrages. Ces problèmes de filtrage et d'unification sont exposés dans la section 8.1.

Une condition peut être soit un fait en tant que tel (elle est alors vérifiée s'il existe un élément connu lui correspondant), soit un filtre, soit un opérateur avec ses arguments. L'argument peut être, selon le cas, un fait ou une expression Lisp. On peut évaluer des procédures en cours de filtrage, créer une variable et affecter à cette nouvelle variable une valeur et on peut ensuite se servir de cette variable.

Il est possible de réaliser des opérations logiques (négation, conjonction) sur des filtres ou éléments de filtres: par exemple l'opérateut **non** oblige à vérifier qu'aucun fait de la base n'est compatible avec l'élément de filtre qui suit.

Les opérateurs apparaissant en partie condition sont:

- **execute** ou **avec** <expression>: évalue <expression> (S-expression Lisp) dans l'environnement, si le résultat de la dernière évaluation est différent de "NIL", on continue, sinon on retourne au noeud de choix le plus récemment visité.
- **enrichir** < variable expression>: pour enrichir l'environnement de filtrage d'une nouvelle liaison <variable valeur>.
- **non** <filtre>: cette condition n'est satisfaite que s'il n'existe pas d'élément de la base de faits qui corresponde à <filtre>.

Langages pour conclusion

Les effets de déclenchement d'une règle sont des transformations de la base des faits, spécialement dans notre cas pour avoir des nouvelles données transformées d'une échelle initiale à une nouvelle échelle dérivée finale en vidant la partition de base de conflits, c'est à dire que tous les conflits seront résolus.

On distingue trois sortes d'effets, en faisant un parallèle avec les trois classes d'instructions de base des langages de programmation:

- instructions de traitement: effets sur la base des connaissances et au premier chef sur la base des faits. On a des primitives pour insérer de nouveaux faits dans la base de faits ou supprimer des faits, donc le moteur est non monotone. Le moteur ne donne aucun changement sur la base de faits en cas d'ajout d'un fait déjà présent dans la base, ou de suppression d'un fait non présent dans la base.

  Pas de primitives pour modifier la base de règles, mais chaque changement d'états de la base de faits change également les états des règles dont le déclencheur contient un filtre qui correspond le prédicat du fait d'ajout ou de suppression. Par conséquent, l'état de la base de règles valides change.

- instructions d'entrée-sortie: communication externe. On propose également la primitive **executer** pour expédier des messages, consulter l'environnement, ou lancer des processus externes au moteur d'inférence et à la base de connaissances.

- instructions de contrôle du moteur lui-même: des primitives peuvent agir sur le comportement du moteur d'inférence. Par exemple: abandon du cycle de base en cours, arrêt du moteur.

Les opérateurs sur les actions en conclusion sont:

1. **ajoute** <filtre>: permet d'ajouter dans la base de faits le <filtre> totalement instancié.
2. **conclut** <filtre>: donne la preuve d'un fait sans insertion dans la base de faits.
3. **supprime** <filtre>: supprime de la base de faits l'ensemble de faits filtrables par <filtre>. Ce système est donc non monotone.
4. **execute** <expression>: évalue une (S-expression Lisp) dans l'environnement. etc.

De même, on définit la classe de règles: principalement, la classe de règles a les champs suivants:

nom : une chaine de caraactères, comme R1, Raction1, etc.

nb-prédicats : une liste de doublets dont le premier terme est un nom de prédicat apparaissant en prémisse de la règle, et le deuxième, le nombre d'occurrences de ce prédicat en prémisse de la règle.

Une prémisse positive ne contient pas les actions **non**, **avec**, **execute**, **echec**, dans ce cas le deuxième terme aura une valeure positive.

Une prémisse négative est une action préfixé par **non**; dans ce cas, le deuxième terme aura une valeur négative, la valeur en général est -1.

nb-prédicats-inconnus : est utilisé pour gérer la sélection des règles candidates, il désigne le nombre de prédicats en partie antécédent n'ayant pas d'instance (lorsque celui-ci prend la valeur 0 la règle est considérée comme candidate), la valeur est un entier.

utilisée? : indique si elle a déjà subi ou non un déclenchement, la valeur est `T` ou `nil`.

antécédents : contient la liste de conditions à vérifier.

conséquent : contient la liste des conclusions de la règle.

Pour toute règle déclenchée le nombre de prédicats n'ayant pas d'instances (le champ **nb-prédicats-inconnus**) est affecté à 1. En effet, seule une nouvelle instance d'un des prédicats comme prémisse positive de la partie **antécédent** permet à nouveau son application. Une suppression d'une dernière instance d'un prédicat comme prémisse négative de la partie antécédent d'une règle permet de rendre cette règle candidate. L'acquisition d'une instance de prédicat permettra d'affecter pour toute règle déjà utilisée, et où il apparaît en partie antécédents, son champ **nb-prédicat-inconnu** à 0 rendant celle-ci candidate.

On communique avec les règles en envoyant des messages pour obtenir des informations.

L'implantation interne d'une règle se trouve à la section A.1.

Les effets des actions dynamiques: **ajout** et **suppression**, sont décrits dans les tableaux 7.2 et 7.1.

7.4.4 L'organisation des règles

Comme les informations, valeurs numériques ou littérales, procédures, une règle de production est représentée sous forme d'*objet* dans la base de connaissances. Elles sont groupées en une ou plusieurs classes. La gestion des règles est ainsi facilitée. Les règles sont aussi séparées en sous-ensembles liés à des sous-problèmes. On peut alors associer un ensemble de règles à une classe particulière des objets dont les instances peuvent l'évoquer, soit à l'intérieur d'une méthode, soit quand une des variables d'instances prend une valeur particulière. Cette hiérarchisation de la base des règles permet aussi de limiter l'espace de recherche des règles.

L'organisation virtuelle des règles se montre à la figure 7.4 de la page 147. La figure 7.5, qui détaille les relations hiérarchiques, se trouve à la page 150, mais dans la base des règles, elles sont rentrées en vrac.

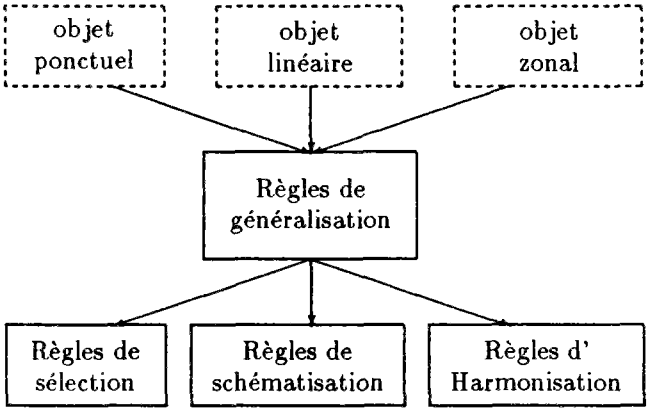


Figure 7.4: Structuration des règles

L'**ajout** d'un fait:

Si : le fait est déjà connu dans la base de faits du cycle précédent

Alors : on ne fait rien

Sinon : on execute le processus suivant:

- ▸ on incrémente le nombre de succès de la règle en cours du declenchement.
- ▸ on actualise le dictionnaire des prédicats.
- ▸ on ajoute ce fait dans le champ `Lfaits-total` du prédicat correspondant.
- ▸ on ajoute le fait dans la base de faits.
- ▸ pour chaque règle où le prédicat du fait est présent comme une prémisse négative (précédé par **non**):

  Si : c'est la première instance qu'on associe au prédicat de ce fait:

  Alors :

  Si : la règle a déjà été déclenchée.

  Alors : on la laisse bloquée: indéclenchable: utilisée(t).

  Sinon : on incrémente de 1 le nb-prédicats-inconnus

  Sinon : on ne fait rien.
- ▸ pour chaque règle où le prédicat du fait est prémisse positive:

  Si : la règle a déjà été déclenchée

  Alors :

  - rendre la règle à nouveau declenchable: "non déclenchée"
  - mettre le nb-prédicat-inconnu à 0 afin de la rendre candidate

  Sinon :

  Si : le nombre de faits connus attachés au prédicat de ce fait ajouté est plus grand ou égal au besoin (de ce fait) de la règle pour pouvoir la declencher

  Alors : on ne fait rien.

  Sinon :

  on décrémente le nb-prédicats-inconnus de la règle. Si ce nombre devient égal à 0, la règle est candidate.

Tableau 7.1: Ajout d'un fait

La **suppression** d'un fait:

Si : le fait n'est pas connu dans la base de faits du cycle précédent

Alors : on ne fait rien

Sinon : on execute le processus suivant:

- on retire ce fait du champ `Lfaits-total` du prédicat du fait
- on retire le fait de la base de faits
- pour chaque règle où le fait est une prémisse positive:

  Si : il y a moins de faits connus associés au prédicat de ce fait que le besoin (de ce prédicat) de cette règle:

  Alors :

  Si : la règle a déjà été déclenchée

  Alors : on la marque non utilisée (le nb-prédicats-inconnu en ce moment precis est 1).

  Sinon : on incrémente de 1 le nb-prédicats-inconnus

  Sinon : on ne fait rien.
- pour chaque règle où le fait est une prémisse négative (prefiixé de "NON"):

  Si : il n'y a plus de faits connus associés au prédicat de ce fait:

  Alors :

  Si : la règle a déjà été déclenchée.

  Alors :

  - on la marque à nouveau déclenchable: non utilisée.
  - on decrémente de 1 le nb-prédicats-inconnus (il devient 0 en ce moment)

  Sinon : si le nb-prédicats-inconnus est plus grand ou égal à 1, on decrémente de 1 le nb-prédicats-inconnus

  Sinon : on ne fait rien.

Tableau 7.2: Suppression d'un fait

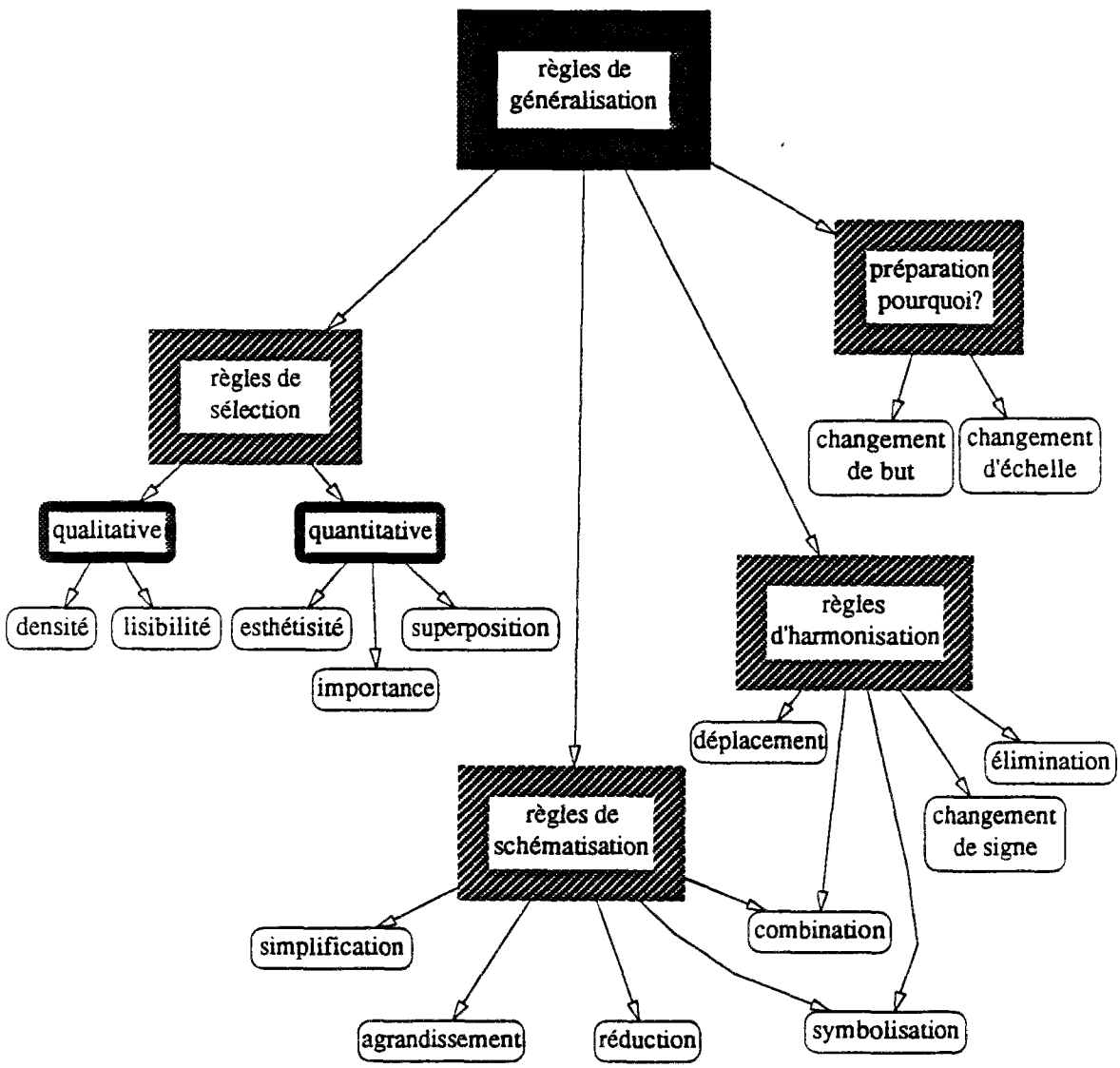


Figure 7.5: Toutes les règles de la G.C.

Chapitre 8

Un petit moteur d'inférences

8.1 Le Filtrage et l'unification

Lorsque le système cherche à appliquer une règle, il est amené à instancier les prémisses de celle-ci. L'instanciation d'une condition avec un élément de la base de faits est effectuée par **filtrage**.

L'étape de filtrage a pour but de déterminer l'ensemble de "conflits", c'est à dire l'ensemble des règles dont les déclencheurs sont compatibles avec l'état courant de la base des faits.

Le filtrage est le mécanisme de base des systèmes avec variables. Il est principalement utilisé pour:

- savoir si une prémisse est élément de la base de faits,
- savoir si une règle est candidate,
- déclencher une règle si elle déclenchable.

L'opération de filtrage consiste à comparer deux expressions symboliques: une *donnée* et un *filtre*.

- la donnée est une structure dont les éléments sont des constantes, par exemple: (**écarter la route A4 de la rivière Marne**)
- Le filtre est une structure qui peut contenir des variables ou des constantes, par exemple: (**écarter la route A4 de la rivière Marne**), dans ce cas le filtrage vérifie l'exacte correspondance de tous les éléments de deux expressions.

  donc comparer un filtre et une donnée:

donnée	=	(écarter la route A4 de la rivière Marne)
filtre	=	(écarter la route A4 de la rivière Marne)

donne succès, et avec

donnée	=	(écarter la route A4 de la rivière **Marne**)
filtre	=	(écarter la route A4 de la rivière **Seine**)

donne un échec.

En général, un filtre contient des symboles spéciaux, des constantes et des variables, par exemple (**écarter la route ?x de la rivière ?y**), où ?x et ?y sont des variables; au travers de tous éléments, on recherche les éléments correspondants pour rendre les deux expressions *filtre* et *donnée* semblables.

Dans notre système avec variables, pour permettre une écriture plus souple des connaissances dans les règles, on utilise plusieurs variables de filtre. Un filtre peut contenir:

- les variables précédées de "?" capture l'élément simple associé de la donnée:

  ?x : lier la variable ?x (et non x,) à un élément simple: un symbole;

  ?- : indique un élément atomique quelconque, il est un *opérateur absorbant atomique*, cet opérateur peut apparaître n'importe où dans un filtre comme un élément atomique, il est compatible avec n'importe quel élément atomique.

  par exemple:

donnée	=	(écarter la route **A4** de la rivière Marne)
filtre	=	(écarter la route **?x** de la rivière Marne)

  donne la valeur **A4** à la variable simple **?x**: [?x : A4].

donnée	=	(écarter la route A4 de la rivière Marne)
filtre	=	(écarter la ?- ?x de la rivière Marne)

  où ?- indique qu"un élément correspondant doit être présent dans la donnée, ici **route**, et la variable simple **?x** a la même valeur **A4**.

- les variables précédées de "&" capture un segment de liste éventuellement vide:

  &x : lie la variable &x à une liste éventuellement vide d'éléments;

  &- : indique une liste quelconque éventuellement vide d'éléments, c'est un *opérateur absorbant de liste*, il peut apparaître n'importe où dans un filtre, il est compatible avec n'importe quelle suite de symboles, éventuellement vide.

  par exemple:

donnée	=	(écarter la route A4 de la rivière Marne)
filtre	=	(écarter &y)

  donne la valeur (*la route A4 de la rivière Marne*) à la variable simple **&y**: [**&y**:(*la route A4 de la rivière Marne*)].

- **avec** ou /: ces deux signes précèdent une expression **Lisp**. Puisque le système a été écrit en Lisp, on a ajouté la possibilité d'avoir des condition **Lisp** à l'intérieur d'un filtre, on peut évaluer des procédures en cours de filtrage.

La condition précédée par "**avec**" ou "/" dans un filtrage est évaluée dans l'environnement local créé lors de l'unification de la règle. Si le résultat de cette évaluation est différente de "Nil", le filtre réussit, sinon il échoue. Le filtrage de (ecarter ?x &x) avec les données (ecarter ca1 ca2) échoue, tandis avec (ecarter c2 (c3 c4)) réussit par lier ?x à c2, &x à (c3 c4).

Aucune restriction syntaxique n'est faite quant aux places des variables simples nommés. On peut définir des concepts aussi variés que: (carrefour C2 Grande-circulation) pour dire que le carrefour C2 est un noeud de grande circulation du monde, ou pour affirmer que ?x intersecte ?y par le filtre (?x ecarter ?y) ou par (ecarter ?x ?y) . On pourrait donc définir le concept de "ecarter" par (ecarter ?x ?y) ou (?x ecarter ?y).

Le filtrage n'a aucune propriété sémantique, seul le concepteur ou l'utilisateur ou l'expert est concerné par des écrits (sous réserve d'une interprétation adéquate) : (?Relation ?obj1 ?obj2) ou (ecarter ?x ?y ?-), etc.

Notons que l'opération de *filtrage* est une opération déterministe et séquentielle. Théoriquement, les variables segments **&-** et **&x** peuvent prendre des séquences variables des éléments. Pour cette raison, on se contente du premier ensemble de valeurs trouvées pour rendre le filtre identique à la donnée, et les variables segments(**&-** et **&x**) sont substituées par une séquence minimale d'éléments. Par exemple la comparaison entre

donnée	=	(écarter la route **A4** de la route **R4**)
filtre	=	(**&x** la route **&y**)

donnera les substitutions:

&x $\Leftarrow$ (écarter)
&y $\Leftarrow$ (**A4** de la route **R4**)

et non pas

&x $\not\Leftarrow$ (écarter la route **A4** de)
&y $\not\Leftarrow$ (**R4**)

Le filtrage est une opération de *semi-unification*, c'est à dire que l'expression mise en correspondance avec le filtre ne contient pas de variable. L'**unification**, quant à elle, autorise la comparaison de deux expressions acceptant des variables. On trouvera dans la section A.2 un algorithme de filtrage programmé en **Lisp** pour le besoin du chaînage avant.

8.2 Avantages et inconvénients

L'avantage d'utiliser la méthode de filtrage classique est de ne manipuler que des termes symboliques sans se préoccuper de la signification des termes utilisés. Cette opération utilisée pour mettre en correspondance une condition d'une règle et un élément de la base de faits a les avantages suivants:

- les symboles spéciaux: `?x`, `?-`, `&-`, `&y` apportent une grande souplesse dans l'écriture des connaissances.
- pas de restriction syntaxique sur un fait, et par conséquent, pas de restriction syntaxique sur les prémisses des règles, sous réserve d'une interprétation adéquate d'une règle, et des correspondances entres les prémisses des règles et les faits de la base de faits.
- pas de restriction syntaxique aux places des variables dans les `filtres`, donc il est facile d'étendre la puissance d'expression de la connaissance.

L'inconvenient est que la structure de contrôle par filtrage lors de la sélection et du déclenchement des règles dans un système avec variables en mode chaînage avant est souvent très coûteuse en temps.

- une condition est filtrée autant de fois qu'elle apparaît dans des règles différentes.
- les conditions d'une règles sont comparées avec tous les éléments de la base de travail de faits.

  Pour une base de connaissance importante, le filtrage seul est donc inapproprié.

De toute évidence, il est indispensable d'allier aux possibilités du filtrage classique, des méthodes efficaces de sélection et de déclenchement des règles.

8.3 Résolution

Pour éviter les inconvénients du filtrage classique, on doit:

- pour une prémisse donnée, avoir accès à l'ensemble des règles où elle apparaît,
- pour un filtre d'une règle, dans les prémisses ou les conclusions, connaître l'ensemble des éléments de la base de faits qui le vérifient,
- connaître les liens qui peuvent exister entre les différentes conditions d'une même règle.

Afin d'augmenter la rapidité d'exécution pour instancier les variables des conditions d'une règles lors du filtrage, on utilise diverses informations visant à réduire le temps de calcul à chaque cycle:

- un prédicat est un objet, chaque prédicat est doté de l'ensemble des règles où il figure en partie prémisse et en partie conclusion,
- chaque condition est dotée, par l'intermédiaire de son prédicat, de l'ensemble des éléments de la base de faits qui la vérifient,
- pour chaque règle, qui est un objet, des indicateurs (variables d'instance) indiquent si elle est candidate, si elle est déjà utilisée dans le cycle courant, si elle est encore potentiellement déclenchable.

les prémisses ne sont pas filtrées avec l'ensemble des faits de la base de faits mais seulement avec ceux apparaisant sur le champ **Lfaits-total** de leur prédicat. Donc, lorsque le système visite une prémisse dont le prédicat est une variable, celle-ci doit avoir déjà été instanciée. On considére qu'un prédicat est le premier élément d'un fait, d'un filtre en général; donc le premier antécédent ne pourra jamais comporter de prédicat variable.

Exemple: soit la règle symétrie-1

```
Si        (symetrique ?R )
      et  (?R ?x ?y)
Alors     ajoute (?R ?y ?x)
```

Soit la base de faits:

```
(Voisin c1 c2)
(intersecter c2 c3)
(symetrique intersecter)
```

L'unification de la première prémisse de la règle ne sera tentée qu'avec (**symetrique intersecter**), tous les autres étant ignorés.

En effet, dans la première règle l'unification est réalisée d'abord avec la première prémisse (**symetrique ?R**) et les faits figurant sur le champ **Lfaits-total** du prédicat **symetrique**. La variable **?R** est donc instanciée par la valeur **intersecter**, la deuxième prémisse pouvant alors être unifiée avec les faits présents sur **Lfaits-total** du prédicat **?R**, auquel est à ce moment substituée par sa valeur **intersecter**.

Si les prémisses de la règle symétrie-1 sont interverties, celle-ci ne sera pas déclenchée correctement.

```
Si        (?R ?x ?y)
      et  (symetrique ?R )
Alors     ajoute (?R ?y ?x)
```

En effet, lorsque le système tente d'unifier la première prémisse, il décèle la présence d'une variable comme prédicat: cette variable n'est pas un objet, sans valeur, le système cherche donc sa valeur dans l'environnement et ne la trouve pas.

Le moteur d'inférence interprète les connaissances comme un système avec variables, non monotone. La démarche du MI en mode chaînage avant est assez simple. Il y a balayage de l'ensemble des règle candidates tant que l'on peut obtenir de nouvelles informations; une règle est candidate, si à chacune de ses prémisses correspond un élément de la base de faits (son champ **nb-prédicats-inconnus** valant alors 0, voir l'annexe A.1 pour la représentation interne d'une règle). Après son application le champ **nb-prédicats-inconnus** est initialisé à 1, ceci pour la rendre à nouveau candidate dès l'acquisition d'une nouvelle instance d'un prédicat de sa partie condition (en admettant, bien sûr, qu'aucun des prédicats de sa partie condition n'a été supprimé). Toute règle déclenchée redevient candidate, si l'un de ses prédicats apparaissant en partie condition obtient une nouvelle instance. Les règles candidates sont visitées par le système dans l'ordre de leur numérotation, de même pour les prémisses.

Une règle sera déclenchée lorsque tous ses antécédents auront été instanciés par des éléments de la base de faits. Les actions figurant dans la partie conclusions de

la règle sont alors envisagés en substituant aux variables des termes par leur valeur enregistrée lors du filtrage.

Le système présenté ici ne prend pas en considération le problème du choix de la règle à appliquer, et ne déploie pas de stratégie sophistiquée lui permettant de minimiser au maximum le nombre d'instanciations inutiles (notons que la création de l'ensemble des règles candidates et la modularisation des règles selon les domaines donnent déjà des performances appréciables).

Le moteur d'inférence est aussi un objet d'une classe défini MI qui a les champs:

- **bdr:** (la base des règles).

- **bdfi:** (la base des données initiales).

- **bdff:** (la base des données finales dérivés de la base des données initiales).

Le moteur d'inférences a des procédures suivantes:

Initialiser: (initialise le moteur en initialisant la base des règles, la base de faits (bdfi, bdff)).

Afficher: (affiche la règle que le moteur execute.)

Editer: (édite des connaissances.)

En-avant: (met le moteur en mode de chaînage-avant.)

etc.

8.4 Chaînage avant

L'algorithme du cycle de base:
phase de détection et de résolution de conflit:
(1)- On prend la première règle candidate
phase de déduction
- On sature sur la règle:
Si: on a obtenu au moins un succès
(c'est à dire un fait nouveau)
Alors: on recommence en (1) avec la nouvelle base de règles candidates
Sinon: -on passe à la règle candidate suivante
-on s'arrete lorsqu'aucune règle candidate n'a pu s'appliquer

Saturer sur une règle signifie obtenir de celle-ci le maximum d'informations, c'est-à-dire la déclencher autant de fois que possible au cours d'un cycle. On a préféré cette solution à celle qui consiste à appliquer une seule fois la règle retenue, ou encore à déclencher (en parallèle) toutes les règles candidates, puis à passer au cycle de base suivant. Lorsqu'une règle est déclenchée, sa partie <conclusions> est évaluée, les différences actions (ajoute, suppression, etc.) qui y figurent sont exécutées.

8.5 Mode d'inférence

D'une manière générale, chaque cycle de base d'un moteur réalise principalement une dérivation de la forme de type **modus ponens**[1]. Néanmoins, nos règles contiennent des actions, pas simplement des faits; la base des faits peut être modifiée à tout moment. Le parallélisme entre exécution d'un cycle de moteur d'inférence et application du modus ponens à deux formules logiques est indépendant du type de chaînage, avant ou arrière.

Les variables dans les règles peuvent être quantifiées universellement, l'opération de filtrage entre déclencheur de règles et faits, inhérente aussi bien au chaînage avant (à partir des faits établis) qu'au chaînage arrière (faits à établir) correspond, en principe, au mode d'inférence appelé spécialisation universelle.

8.6 Stratégie du développement de la recherche

Le mode d'invocation des règles (chaînage avant ou arrière) et les modes d'inférences, n'ont qu'une incidence partielle sur la détermination des enchaînements de règles. On superpose la stratégie *en profondeur d'abord* pour réduire et ordonner les choix des règles, tant qu'il y l'ajout d'un fait nouveau, le moteur intégre immédiatement les conclusions et commence un nouveau cycle avec une nouvelle base de règles qui tient compte du nouvel état de la base de faits résultant de cat ajout. Sinon, le moteur cherche à saturer toute la base de règles candidates au cours du même cycle.

8.7 Non monotonie

Le système fonctionne en sens non monotone: tous les faits établis sont susceptibles d'être retirés de la base, un fait peut être remis en cause.

On a bien des situations qui conduisent à de telles suppresssion:

- en raison de la suppression d'objets:
  - Un objet supprimé par une règle de suppression pourrait être sauvé par une autre règle de priorité plus grande,
  - Un objet qui était obligé précédemment de se déplacer en raison de la superposition avec un autre objet peut se trouver tout seul après la suppression de l'autre objet par une autre règle.
- en raison du regroupement des actions,
- en raison des règles par défaut dont les conclusions sont à réviser. Par exemple, la régle *si on n'a pas de choix, on supprime l'objet faute de place disponible*, et la règle *tant que c'est possible, on garde tout objet* sont vues comme des règles par défaut dès lors qu'on n'est pas certain de la décision.

[1] Voir la section 3.3

8.8 Fonctionnement sur un exemple

Soit ΣC: C_1 , C_2, ..., C_n sont les carrefours. Il s'agit de déterminer s'ils sont tous dans les limites du seuil de séparation; si non, il faut séparer l'un de l'autre, ou séparer l'un d'un sous ensemble de ΣC.

La figure 2.4 de la page 37 montre la carte à l'échelle initiale 1:25 000, avec les réductions sans généralisation à différentes échelles.

La figure 9.3 de la page 172 montre la carte réduite à l'échelle 1:100 000 sans généralisation, avec les signes conventionnels de cette échelle.

La base de règles:

```
(Rg9.1 Rg2.1 Rg2.2 Rg3.1 Rg3.2
 Rg3.3 Rg8.1 Rg8.2 Rg8.3 Rg8.6
 Rg8.4 Rg8.5 Rg4.1 Rg4.2 Rg4.3
 Rg4.4 Rg4.5 Rg6.4 Rg6.1)
```

Elles sont dans l'annexe C

Ci-dessous est un échantillon de processus du moteur d'inférence:

```
echelle initiale : 1/25 000 ;
echelle finale : 1/100 000 ;
reduction e1/e2 : .25

****AVANT3***AVANT3***AVANT3***AVANT3***AVANT3***
**  ON COMMENCE LE MOTEUR EN CHAINAGE AVANT3   **
************************************************


******Initialisation du moteur**********

*La base_de_faits initiale = :
(V-c S1 S2)   (V-c S1 S3)   (V-c S2 S1)
(V-c S2 S3)   (V-c S3 S1)   (V-c S3 S2)
(V-c S3 S21)  (V-c S4 S11)  (V-c S4 S12)
(V-c S4 S19)  (V-c S9 S10)  (V-c S10 S9)
(V-c S11 S4)  (V-c S11 S12) (V-c S12 S4)
(V-c S12 S11) (V-c S12 S19) (V-c S12 S20)
(V-c S13 S26) (V-c S19 S4)  (V-c S19 S12)
(V-c S19 S20) (V-c S20 S12) (V-c S20 S19)
(V-c S21 S3)  (V-c S25 S27) (V-c S26 S13)
(V-c S27 S25) (Voisin V-c)  (Voisin V-d)
(symetrique V-c) (symetrique V-d)
(symetrique Voisin)



******* Le nouveau cycle ***

*La base regles valides = :
      (Rg2.1 Rg8.1 Rg8.4)  **
```

```
avec la Rg2.1
-- Ajout de :
       (Ecarter S25 S27 1.345043)
       (Ecarter S19 S20 5.941544)
       (Ecarter S26 S13 2.841026)
       (Ecarter S20 S12 1.802406)
       (Ecarter S19 S12 .9984245)
       (Ecarter S11 S12 6.278919)
       (Ecarter S4 S12 5.159216)
       (Ecarter S19 S4 1.641409)
       (Ecarter S11 S4 6.469025)
       (Ecarter S21 S3 .418251)
       (Ecarter S2 S3 5.662346)
       (Ecarter S3 S1 5.584221)
       (Ecarter S2 S1 4.579981)
13 succes avec la regle- : Rg2.1
 <<FIN>> de la saturer!

******* Le nouveau cycle ***
*La base regles valides = :
     (Rg3.1 Rg8.1 Rg8.4 Rg6.4)  **

 avec la Rg3.1
-- Ajout de :
      (Ecarter-g S2 (S1 S3) 5.662346)
--SUPPRESSION de:
      (Ecarter S2 S1 4.579981)
--SUPPRESSION de:
      (Ecarter S2 S3 5.662346)
-- Ajout de :
      (Ecarter-g S11 (S4 S12) 6.469025)
--SUPPRESSION de:
      (Ecarter S11 S4 6.469025)
--SUPPRESSION de:
      (Ecarter S11 S12 6.278919)
-- Ajout de :
      (Ecarter-g S19 (S4 S12) 1.641409)
--SUPPRESSION de:
      (Ecarter S19 S4 1.641409)
--SUPPRESSION de:
      (Ecarter S19 S12 .9984245)
3 succes avec la regle- : Rg3.1
 <<FIN>> de la saturer!

******* Le nouveau cycle ***
*La base regles valides = :
    (Rg2.2 Rg3.2 Rg3.3
     Rg8.1 Rg8.4 Rg6.4 Rg6.1)  **

 avec la Rg2.2
-- Ajout de :
```

```
 (Ecarter-g S19 S4 S12 S20 5.941544)
--SUPPRESSION de:
 (Ecarter-g S19 (S4 S12) 1.641409)
1 succes avec la regle- : Rg2.2
 <<FIN>> de la saturer!

******* Le nouveau cycle ***
*La base regles valides = :
   (Rg3.2 Rg3.3 Rg8.1 Rg8.4 Rg6.4 Rg6.1)  **

-on n'a pas eu de succes avec la regle- : Rg3.2

Le reste des regles valides de ce cycle :
       (Rg3.3 Rg8.1 Rg8.4 Rg6.4 Rg6.1)

-on n'a pas eu de succes avec la regle- : Rg3.3

Le reste des regles valides de ce cycle :
 (Rg8.1 Rg8.4 Rg6.4 Rg6.1)


-on n'a pas eu de succes avec la regle- : Rg8.1

Le reste des regles valides de ce cycle :
 (Rg8.4 Rg6.4 Rg6.1)

-on n'a pas eu de succes avec la regle- : Rg8.4

Le reste des regles valides de ce cycle :
 (Rg6.4 Rg6.1)

 avec la Rg6.4
-- Ajout de :
     (Action (Ecarter S3 S1 5.584221))
     (Action (Ecarter S21 S3 .418251))
     (Action (Ecarter S4 S12 5.159216))
     (Action (Ecarter S20 S12 1.802406))
     (Action (Ecarter S26 S13 2.841026))
     (Action (Ecarter S19 S20 5.941544))
     (Action (Ecarter S25 S27 1.345043))
7 succes avec la regle- : Rg6.4  <<FIN>> de la saturer!

******* Le nouveau cycle ***
*La base regles valides = :
   (Rg6.1)  **

avec la Rg6.1
-- Ajout de :
    (Action (Ecarter-g S11 (S4 S12) 6.469025))
    (Action (Ecarter-g S2 (S1 S3) 5.662346))
2 succes avec la regle- : Rg6.1  <<FIN>> de la saturer!
```

```
******* Le nouveau cycle ***
*La base regles valides = : ()  **
Je ne peux rien deduire de plus.


*Les nouvelles connaissances deduites totales   sont:*

 = :((Ecarter S25 S27 1.345043)
     (Ecarter S19 S20 5.941544)
     (Ecarter S26 S13 2.841026)
     (Ecarter S20 S12 1.802406)
     (Ecarter S4 S12 5.159216)
     (Ecarter S21 S3 .418251)
     (Ecarter S3 S1 5.584221)
     (Ecarter-g S2 (S1 S3) 5.662346)
     (Ecarter-g S11 (S4 S12) 6.469025)
     (Ecarter-g S19 S4 S12 S20 5.941544)
     (Action (Ecarter S3 S1 5.584221))
     (Action (Ecarter S21 S3 .418251))
     (Action (Ecarter S4 S12 5.159216))
     (Action (Ecarter S20 S12 1.802406))
     (Action (Ecarter S26 S13 2.841026))
     (Action (Ecarter S19 S20 5.941544))
     (Action (Ecarter S25 S27 1.345043))
     (Action (Ecarter-g S11 (S4 S12) 6.469025))
     (Action (Ecarter-g S2 (S1 S3) 5.662346)))

Et voila , c'est tout

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


 on sort la carte generalisee en code PostScript

La carte a generaliser se trouve au fichier: marmande100F.ps
= marmande100F.ps
?
```

Chapitre 9

Exemples

Dans ce chapitre, nous allons illustrer par des opérations de la GC des exemples de réalisation. Ils sont basés sur la ville de Marmande[1], proposée par Monsieur Weger de l'IGN, et sont des exemples pédagogiques classiques de la généralisation cartographique manuelle à l'IGN. Les généralisations sont réalisées avec le système implanté en Le-Lisp et Ceyx.

9.1 La base des objets du réseau

On va représenter essentiellement le fonctionnement du processus de résolution en entier sur le réseau de la ville de Marmande. La figure 2.4 de la page 37 montre la carte à l'échelle initiale 1:25 000, avec les réductions sans généralisation à différentes échelles. Nous allons ici généraliser cette carte selon différents facteurs de réduction d'échelle en comparant avec les réductions sans généralisation:

Le réseau est composé de:

- un chemin de fer **F5** composé des arcs: (a_{51}, a_{52}, a_{53}, a_{54}, a_{55})
- une rivière La Garonne **H6** composée des arcs: (a_{61}, a_{62}) et (a_{63}, a_{64}, a_{65})
- Le périphérique de la ville: (a_{71}, a_{72}, a_{73}, a_{74}, a_{75})
- une route nationale **N3** composée des arcs: (a_{31}, a_{32}, a_{33}, a_{34}, a_{35}, a_{36}, a_{37})
- des routes départementales:
  - D_1 (a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19})
  - D_{11}: (a_{110}, a_{111}, a_{112}, a_{113}, a_{114}, a_{115})
- des routes régionales:
  - R4: (a_{41}, a_{42}, a_{43})

[1] Les tomates de Marmande sont bonnes et connues, semble-t-il

- R2: (a_{21})
- R8: (a_{81}, a_{82}, a_{83}, a_{84}, a_{85},)
- R9: (a_{91}, a_{92}, a_{93})
- R10: (a_{101}, a_{102}, a_{103})
- R12: (a_{121})
- R13: (a_{12}, a_{13})

▸ des ruelles d'agglomération:

- R13: (a_{131}, a_{132}, a_{133}, a_{134})
- R14: (a_{141})
- R15: (a_{151})

Il se peut que le nom de certains arcs ne figurent pas sur la carte pour des raisons de lisibilité. Mais ils sont tous codés suivant les routes dont ils dépendent.

9.2 Processus de généralisation

La liste de priorité des objets est:

```
(H, F, E, P, A, N, D, R)
```

▸ H: Hydraulique,

▸ F: Voie ferrée,

▸ E: Européeenne,

▸ P: Péripherique,

▸ A: Autoroute,

▸ N: Nationale,

▸ D: Départementale,

▸ R: Régionale

L'algorithme de procédure est le suivant:

▸ Processus de sélection:

- qualitatif,
- quantitatif.

▸ Détection de confusion localement, route par route, arc par arc; s'il y a des confusions, le processus d'écartement local et éventuellement le processus de linéarisation local seront déclenchés.

- Détection des confusions des carrefours.
- Processus d'harmonisation
  - détection de translation globale
  - revenir en arrière
  - des sélections et suppressions supplémentaires.

9.3 De 1:25 000 aux échelle inférieures

9.3.1 De 1:25 000 à 1:50 000

La figure 9.1 de la page 170 montre la carte réduite à l'échelle 1:50 000 sans généralisation, avec les signes conventionnels de cette échelle.

La figure 9.2 de la page 171 montre la carte réduite à l'échelle 1:50 000 avec généralisation, avec les signes conventionnels de cette échelle.

Pour le signe conventionnel de voie ferrée, à grande échelle, 1:25 000, où la place n'est pas aussi limitée, le nombre de traits peut indiquer le nombre de voies tandis l'épaisseur du trait varie selon le gabarit des voies. Dés l'échelle 1:50 000, cette solution conduirait à un encombrement excessif des voies multiples, on doit abanbdonner ce principe; de plus, les voies ferrées doublant fréquemment des routes, on ne peut les faire apparaître qu'à l'aide de signes adventifs, petits traits perpendiculaires symbolisant les traverses. Une solution plus logique: l'épaisseur du trait est fonction de la largeur de la voie et le nombre de traits perpendiculaires accolés indique sans ambiguïté le nombre de voies.

Etant données la faible densité et l'importance du réseau ferroviaire, cette convention est valable jusqu'aux très petites échelles.

Comme le facteur de réduction est 2, après la réduction, la carte reste lisible, il n'y pas de grosse différence entre les deux cartes, sauf:

- l'arc **a21** de la route **R2** se trouve maintenant sur la Garonne, il est superposé à l'arc **a61**,
- l'arc **a82** de la route **R8** entre le carrefour **S16** et **S18** colle avec l'arc **a52** de la voie ferrée **F5**.

Donc le processus a juste réalisé une procédure d'écartement en linéarisant les arcs de priorité inférieure.

Un échantillon du processus:

```
 pre-traiter1:
ecarter-ps a21 de : a61
ecarter-ps a82 de : a52
ecarter-ps a85 de : a55

 pre-traiter3-sg:
ecarter-arc a82 de : a52
ecarter-arc a122 de : a75
```

```
 on sort la carte generalisee en code PostScript
 echelle initiale : 1/25 000 ;
 echelle finale : 1/50 000 ;
 reduction e1/e2 : .5

La carte a generaliser se trouve au fichier:
   marmande50F.ps
= marmande50F.ps
?
```

9.3.2 De 1:25 000 à 1:100 000

La figure 9.3 de la page 172 montre la carte réduite à l'échelle 1:100 000 sans généralisation, avec les signes conventionnels de cette échelle.

La figure 9.4 de la page 173 montre la carte réduite à l'échelle 1:100 000 avec généralisation , avec les signes conventionnels de cette échelle.

Comme le facteur de réduction est 4, après la réduction, il y a plusieurs endroits où les objets se superposent l'un à l'autre:

- l'arc a21 de la route R2 se trouve maintenant de plus en plus sur la Garonne, il est superposé à l'arc a61,
- l'arc a82 de la route R8 entre le carrefour S16 et S18 est presque caché, avec son élargissement de signe conventionnel; l'arc a52 de la voie ferrée F5, et a85 contre a55 de la même façon, même coté.
- l'arc a73 du péripherique P7 se heurte à l'arc a53 de la voie ferrée F5,
- l'arc a134, entre S25 et S26, de la ruelle d'agglomération R13, se colle à l'arc a75 du péripherique P7
- le triangle S1-S2-S3 disparaît à cause de l'élargissement de ses arcs de bord.

Donc le processus a déclenché:

- une procédure d'écartement en linéarisant les arcs de basse priorité pour séparer les arcs collés ensemble.
- une procédure de séparation de carrefours: S2 de S1

```
ecrire la base de faits initiaux dans *bdf.ll*
  La base de faits au fichier bdf.ll :

charger le fichier de base de faits initiaux:  *bdf.ll*

 * Charger le fichier de Base de Regles: B2newBdr.MI1 : *

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 pre-traiter1:
```

```
ecarter-ps a114 de : a71
ecarter-ps a21 de : a61
ecarter-ps a73 de : a53
ecarter-ps a82 de : a52
ecarter-ps a85 de : a55

 pre-traiter3-sg:
ecarter-arc a12 de : a62
ecarter-arc a102 de : a18
ecarter-arc a134 de : a75
ecarter-arc a122 de : a75

 Les actions a effectuer sont les suivantes :
(Action (Ecarter-g S2 (S1 S3) 5.662346))
(Action (Ecarter-g S11 (S4 S12) 6.469025))
(Action (Ecarter S25 S27 1.345043))
(Action (Ecarter S19 S20 5.941544))
(Action (Ecarter S26 S13 2.841026))
(Action (Ecarter S20 S12 1.802406))
(Action (Ecarter S4 S12 5.159216))
(Action (Ecarter S21 S3 .418251))
(Action (Ecarter S3 S1 5.584221))
Et voila , c'est tout

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

 une action :
   ECARTER S2 pour eloigner ses voisins : (S1 S3);
   Direction: 335.7722
 pour avoir la distance: 5.662346

 une action :
 On laisse ces 3 scs tels qu'ils sont:
  S11 et (S4 S12)

   ECARTER S25 pour eloigner ses voisins : (S27);
 Direction: 174.2794
 pour avoir la distance: 1.345043


 une action :
   ECARTER S19 pour eloigner ses voisins : (S20);
  Direction: 44.99999
 pour avoir la distance: 5.9415444

  REVERIFICATION:
S19 est MIS au centre-10
```

```
 une action :
   ECARTER S26 pour eloigner ses voisins : (S13);
  Direction: 158.7495
 pour avoir la distance: 2.841026
 On ne deplace que: 2.559645 au lieu de: 2.841026


 une action :
   ECARTER S20 pour eloigner ses voisins : (S12);
  Direction: 22.38013
 pour avoir la distance: 1.802406

 une action :
IMPOSSIBLE de deplacer: S4
C'est une croix-dix, a translater! translater


   ECARTER S21 pour eloigner ses voisins : (S3);
 Direction: 321.5198
 pour avoir la distance: .418251

   ECARTER S3 pour eloigner ses voisins : (S1);
Direction: 78.2317
 pour avoir la distance: 5.584221

 pre-traiter1:
ecarter-ps a114 de : a71
ecarter-ps a21 de : a61
ecarter-ps a85 de : a55
```

9.3.3 De 1:25 000 à 1:250 000

La figure 9.5 de la page 174 montre la carte réduite à l'échelle 1:250 000 sans généralisation, avec les signes conventionnels de cette échelle.

La figure 9.6 de la page 175 montre la carte réduite à l'échelle 1:250 000 avec généralisation, avec les signes conventionnels de cette échelle.

Dans la généralisation cartographique manuelle, pour aboutir à une carte finale, on essaie d'avoir comme référence une carte d'échelle pas trop éloignée de celle de la carte finale, le facteur ne devant pas dépasser pas 4 ou 5; la raison en est que le cartographe a du mal à évaluer visuellement les différences de caractéristiques de deux cartes, et appécie mal l'espace disponible, par conséquent, il peut avoir tendance à surcharger la carte ou au contaire à la sous-charger, s'il y a trop d'écart entre les deux.

Mais ce genre de problème ne se pose pas pour notre système, quel que soit le rapport d'échelle; la règle de lisibilié est toujours la même numériquement. Comme le facteur de réduction est 10, qui est considéré comme grand dans la généralisation

manuelle, les cartes ont des caractéristiques plus différentes que jamais.

Par définition, la carte est une image conventionnelle qui n'est pas et ne peut être rigoureusement semblable au terrain représenté. Quelle que soit la densité des phénomènes figurés, la possibilité de mesure est conditionnée soit par une exactitude positionnelle (mise en place rigoureuse par rapport au réseau de référence), soit par une exactitude relationnelle (disposition relative des phénomènes l'un par rapport à l'autre). Ces deux qualités de la carte ne coexistent que sur les plans ou sur les cartes à très grande échelle où la part de convention est réduite. Dans le cas général, on réalise un compromis et l'exactitude relationnelle devient prédominante au fur et à mesure que l'échelle décroît. Sur une carte de 1:25 000, il est presque possible de contenir tous les objets comme dans la nature, alors à l'échelle 1: 250 000, c'est plutôt une caricature de la réalité, c'est juste avant l'étape de symbolisation complète des objets: après la réduction, il y a plusieurs endroits où les objets se superposent l'un à l'autre:

- l'arc **a21** de la route **R2** est en train de boire de plus en plus sur la Garonne en superposant sur l'arc **a61**,
- l'arc **a82** de la route **R8** entre le carrefour **S16** et **S18** a presque caché, avec son élargissement de signe conventionnel, l'arc **a52** de la voie ferrée **F5**, et **a85** contre **a55** de la même façon, même coté.
- l'arc **a73** du peripherique **P7** a dépassé l'arc **a53** de la voie ferrée **F5**; il se trouve de l'autre coté et a rejoint l'arc **a83**.
- la ruelle d'agglomération **R13** est entièrement cachée par la périphérique **P7**.
- L'arc **a12** (entre **S1** et **S2**) et l'arc **a115** (entre **S2** et **S3**) sont cachés par l'arc **a70** (entre **S1** et **S3**), ce dernier a été beaucoup plus agrandi que ses 2 voisins. Par conséquent, le triangle **S1-S2-S3** disparaît à cause de l'élargissement de ses arcs de bord.
- De même, on ne distingue plus le carrefour complexe composé de (**S4, S12, S19, S20**)

Donc le processus a eu les effets suivants:

- Par un processus de sélection qualitative, la ruelle d'agglomération est éliminée,
- Par la règle de détection de conflits, l'arc **a21** ne trouve pas assez de place entre ses deux voisins l'arc **a61** et l'arc **a31**, et il n'est pas plus puissant qu'eux, il est disparaît. De même les arcs **a115, a43, a12, a13,**
- Par les règles d'harmonisation, la voie ferrée **F5** est translatée entièrement en l'éloignant du périphérique **P7**. Par la propagation, la route **R8** subit une translation dans la même direction que la voie ferrée pour garder uns distance minimum entre elle et la voie ferrée. En se déplaçant, la route **R8** subit aussi une linéarisation locale (voir la section 6.5.6).
- une procédure d'écartement en linéarisant les arcs de base priorité pour séparer les arcs collés ensemble.
- une procédure de séparation de carrefours: **S2** de **S1**

9.3.4 De 1:25 000 à 1:500 000

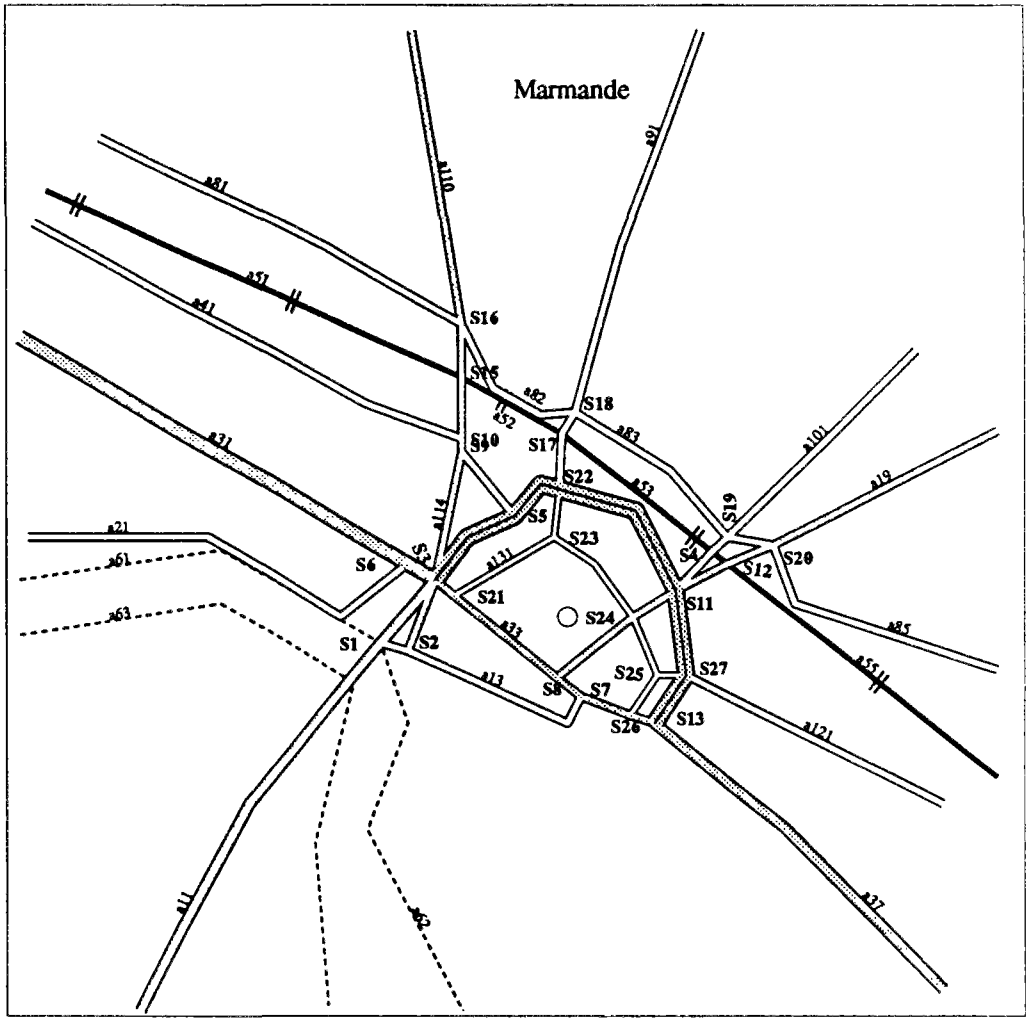
La figure 9.7 de la page 176 montre la carte sans généralisation réduite à l'échelle 1:500 000, avec les signes conventionnels de cette échelle.

La figure 9.8 de la page 177 montre la carte avec généralisation réduite à l'échelle 1:500 000, avec les signes conventionnels de cette échelle,

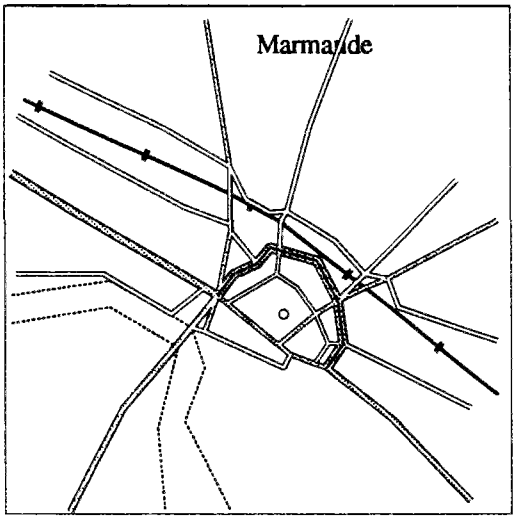
9.3.5 De 1:25 000 à 1:1000 000

La figure 9.9 de la page 178 montre la carte sans généralisation, réduite à l'échelle 1:1000 000, avec les signes conventionnels de cette échelle.

La figure 9.10 de la page 179 montre la carte avec généralisation, réduite à l'échelle 1:1000 000, avec les signes conventionnels de cette échelle.

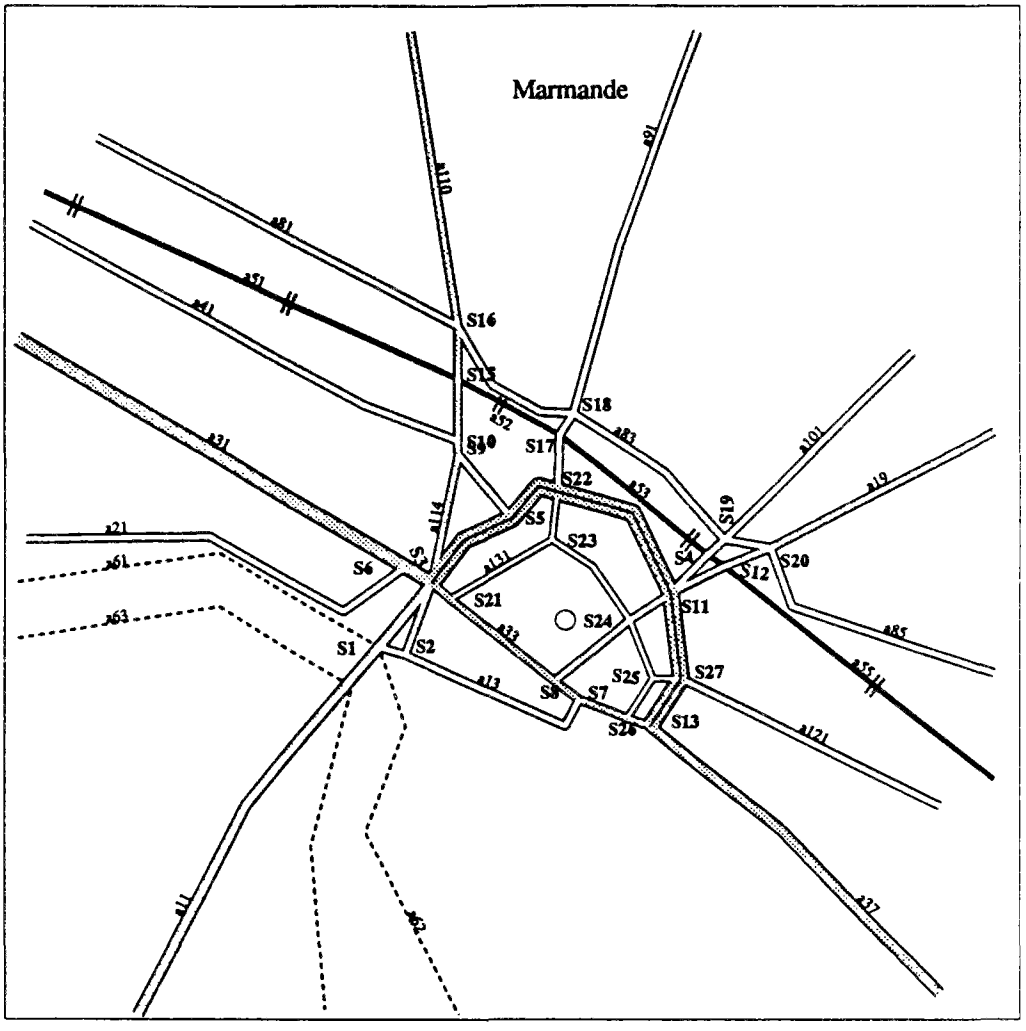


Ci-dessus: echelle finale: 1/50 000 agrandie au 1/25 000 sans generalisation

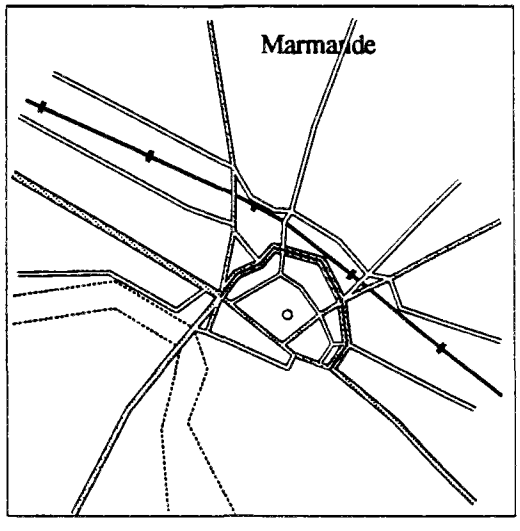


1/50 000

Figure 9.1: En bas: la carte réduite à 1:50 000 sans généralisation; En haut: agrandie à l'échelle 1:25 000

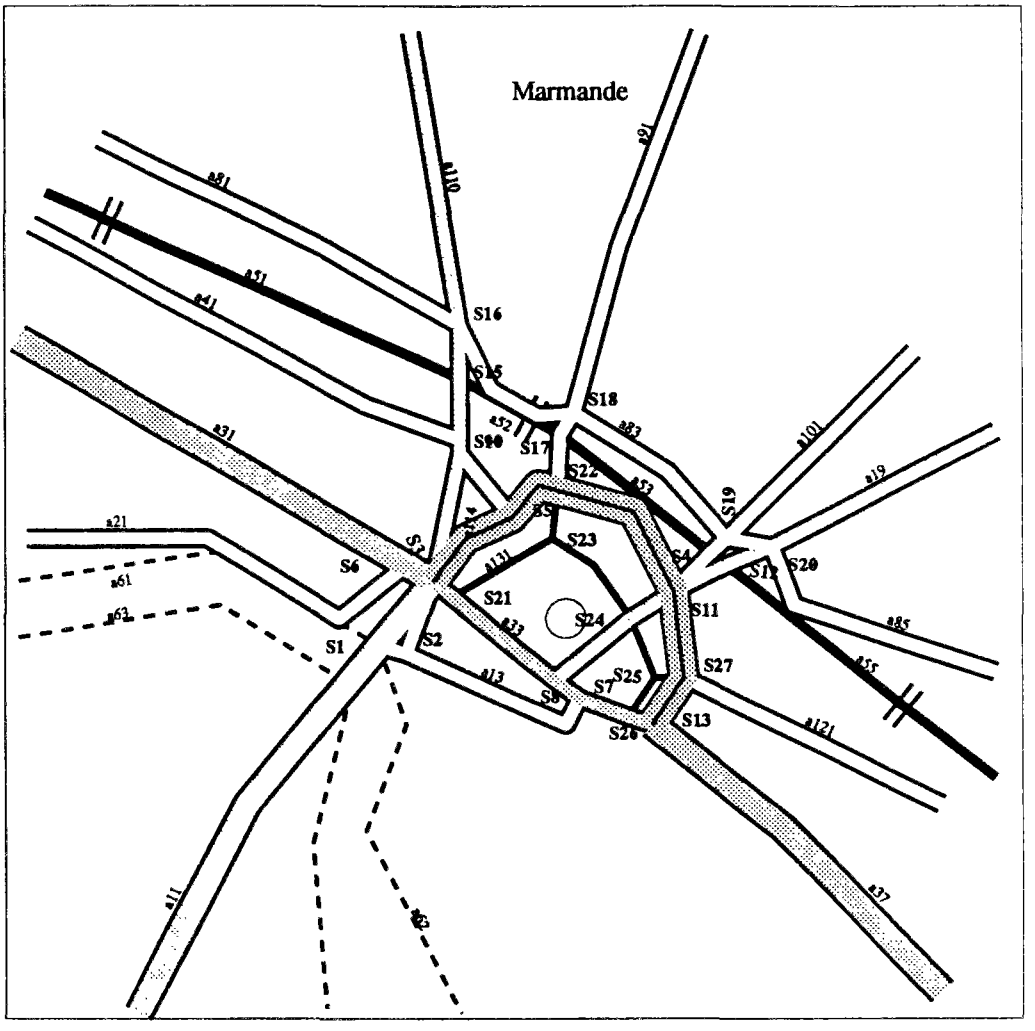


Ci-dessus: echelle finale 1/50 000 agrandie au 1/25 000 avec generalisation



1/50 000

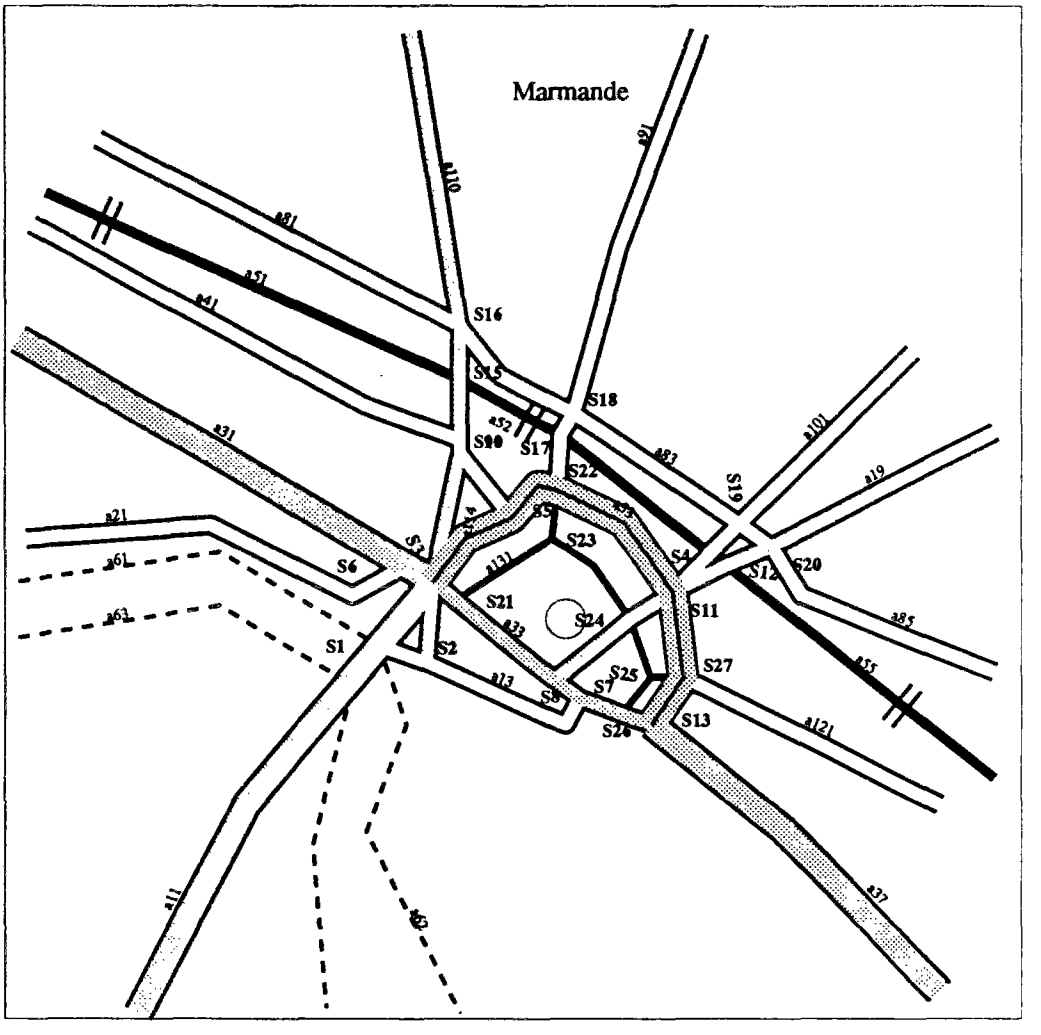
Figure 9.2: En bas: la carte réduite à 1:50 000 avec généralisation;
En haut: agrandie à l'échelle 1:25 000



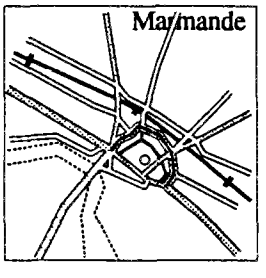
Ci-dessus: echelle finale: 1/100 000 agrandie au 1/25 000 sans generalisation

1/100 000

Figure 9.3: En bas: la carte réduite à 1:100 000 sans généralisation;
En haut: agrandie à l'échelle 1:25 000.

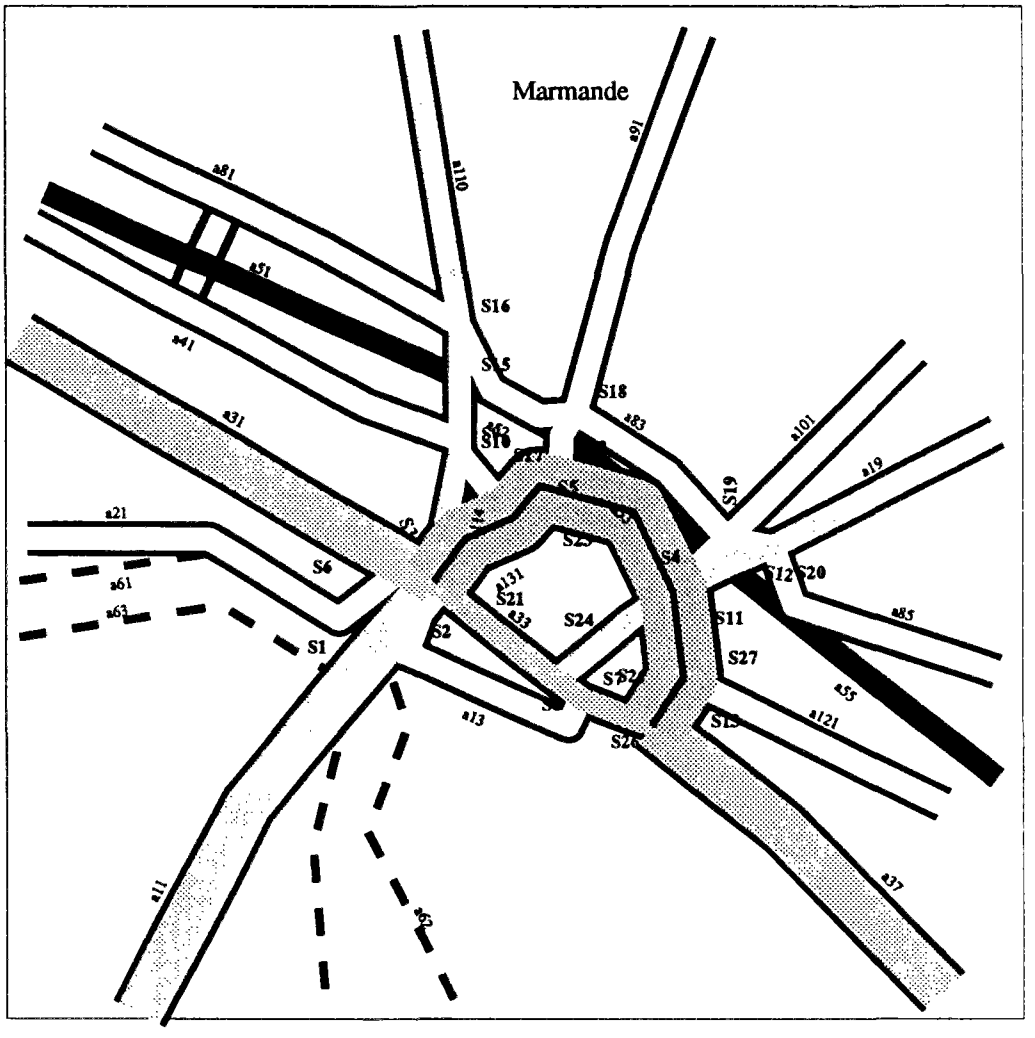


Ci-dessus: echelle finale: 1/100 000 agrandie au 1/25 000 avec generalisation

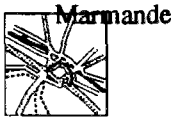


1/100 000

Figure 9.4: En bas: la carte réduite à 1:100 000 avec généralisation;
En haut: agrandie à l'échelle 1:25 000

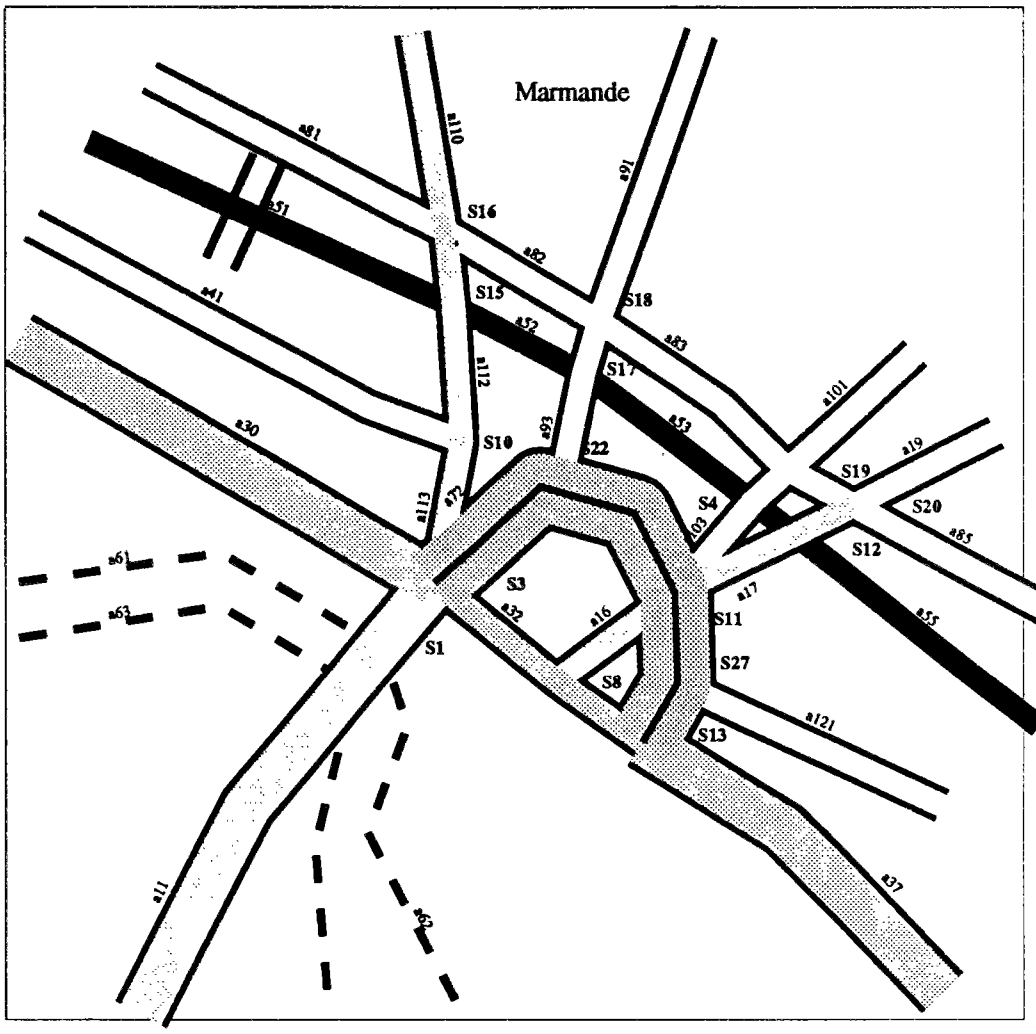


Ci-dessus: echelle finale: 1/250 000 agrandie au 1/25 000 sans generalisation

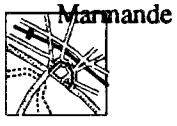


1/250 000

Figure 9.5: En bas: la carte réduite à 1:50 000 sans généralisation;
En haut: agrandie à l'échelle 1:25 000

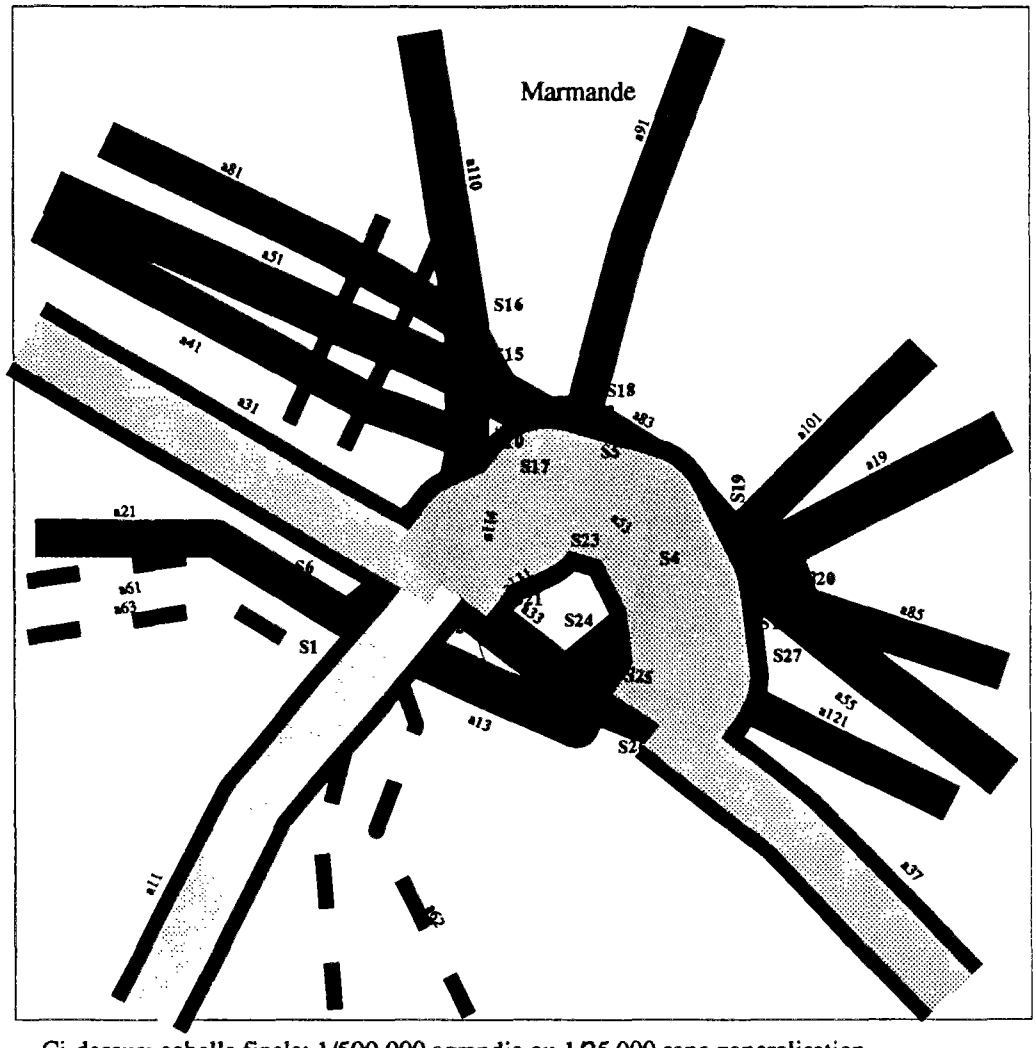


Ci-dessus: echelle finale: 1/250 000 agrandie au 1/25 000 avec generalisation



1/250 000

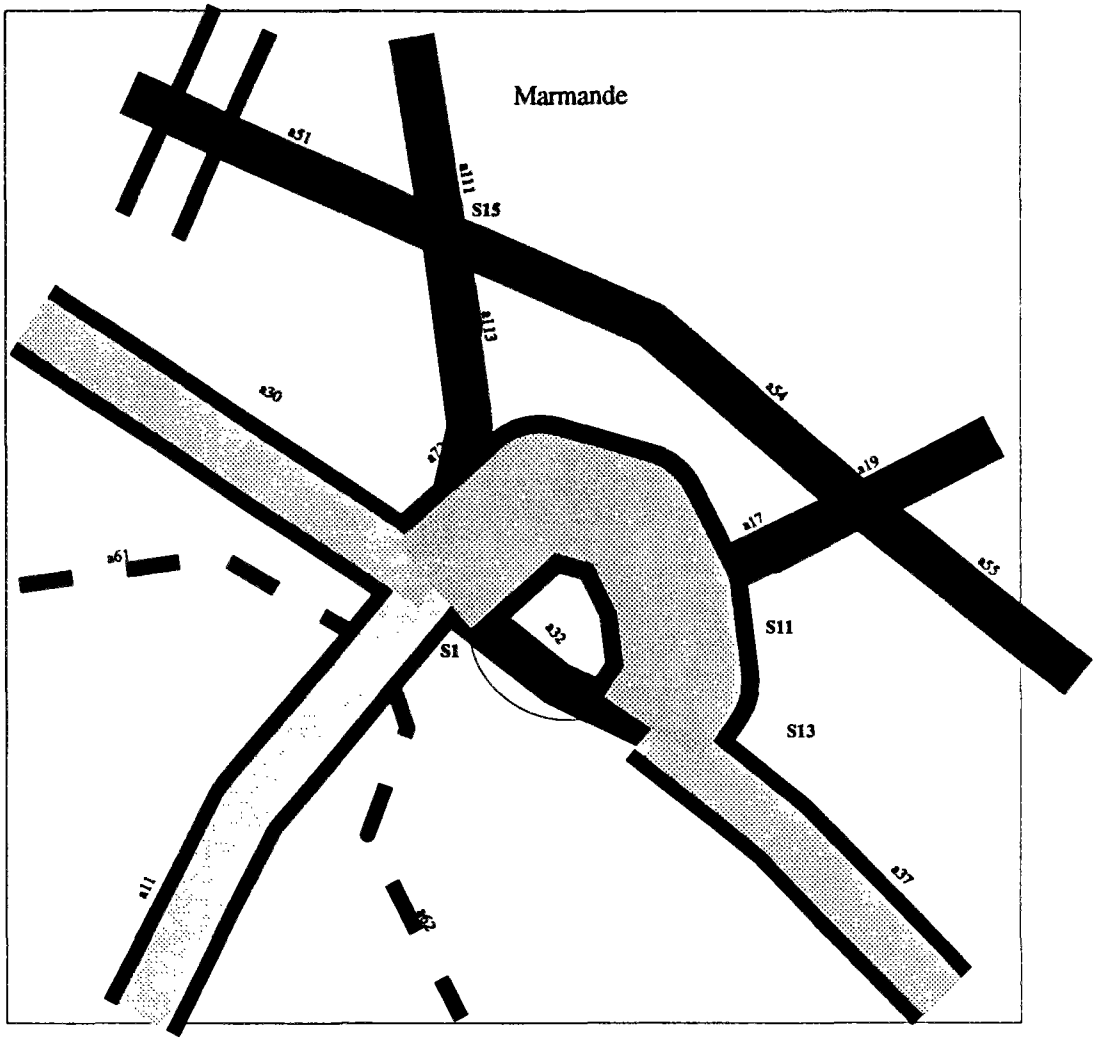
Figure 9.6: En bas: la carte réduite à 1:250 000 avec généralisation;
En haut: agrandie à l'échelle 1:25 000



Ci-dessus: echelle finale: 1/500 000 agrandie au 1/25 000 sans generalisation

1/500 000

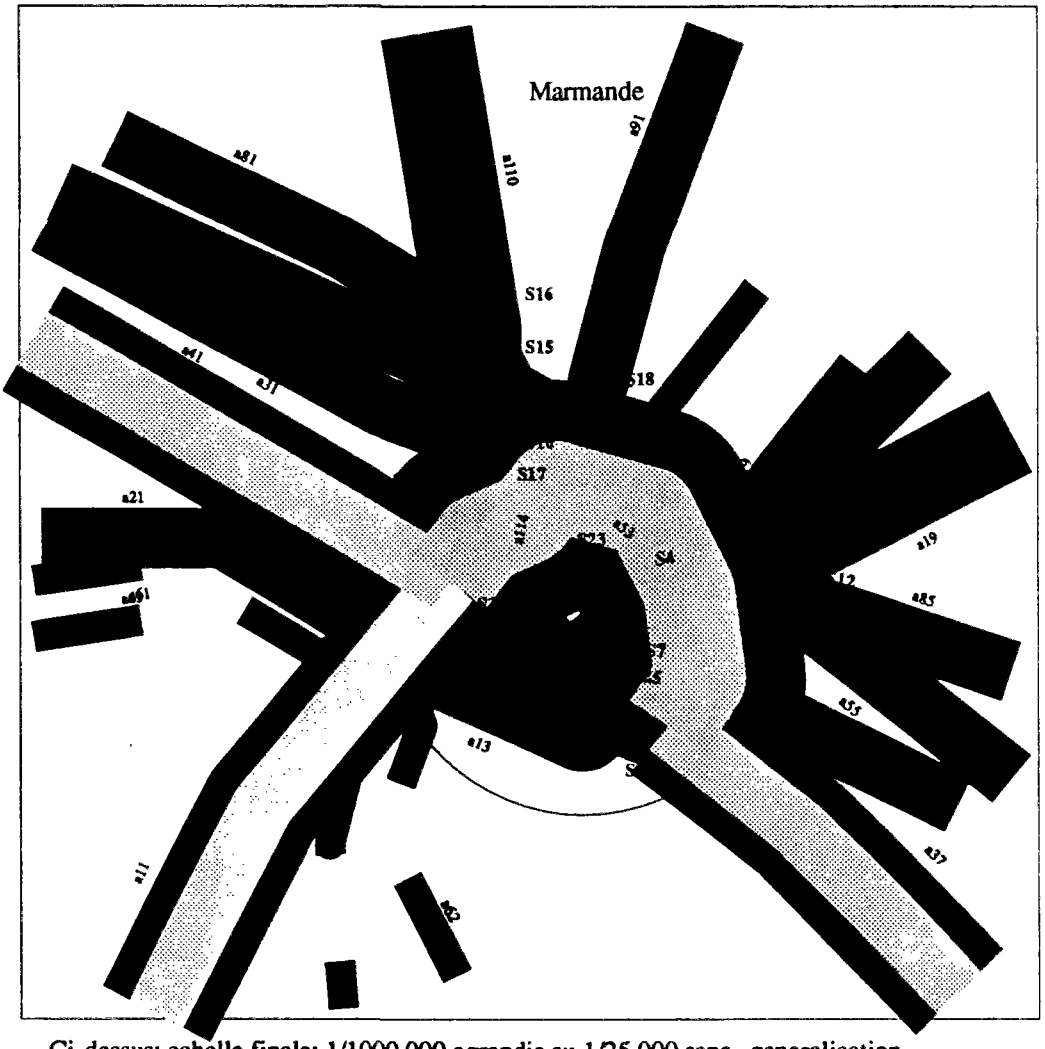
Figure 9.7: En bas: la carte réduite à 1:50 000 sans généralisation;
En haut: agrandie à l'échelle 1:25 000



Ci-dessus: echelle finale: 1/500 000 agrandie a 1/25 000 avec generalisation

1/500 000

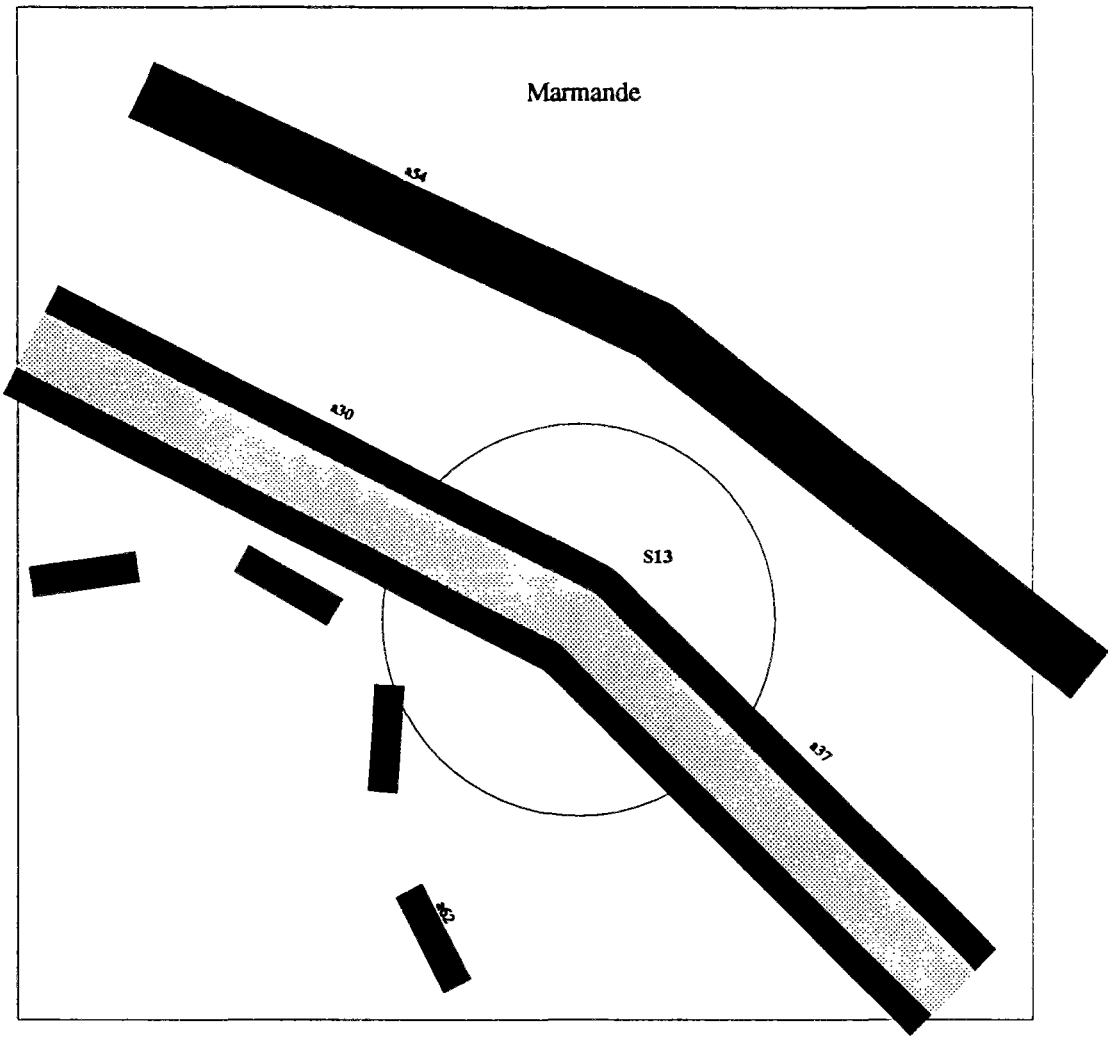
Figure 9.8: En bas: la carte réduite à 1:500 000 avec généralisation;
En haut: agrandie à l'échelle 1:25 000



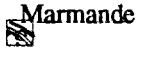
Ci-dessus: echelle finale: 1/1000 000 agrandie au 1/25 000 sans generalisation

1/1000 000

Figure 9.9: En bas: la carte réduite à 1:50 000 sans généralisation;
En haut: agrandie à l'échelle 1:25 000



Ci-dessus: echelle finale: 1/1000 000 agrandie au 1/25 000 avec generalisation



1/1000 000

Figure 9.10: En bas: la carte réduite à 1:1000 000 avec généralisation;
En haut: agrandie à l'échelle 1:25 000

Partie IV

Conclusion générale

Analyse des résultats

Si on regarde les séries de cartes de la ville de Marmande généralisées à partir d'une carte à l'échelle 1/25 000, cartes pour lesquelles les règles suggérées et citées jusqu'à maintenant sont appliquées de façon systématique, on constate les résultats suivants: à l'échelle 1/100 000, les tracés sont fiables par rapport à l'échelle initiale 1/25 000, puisqu'il y a que des petits décalages, le facteur de réduction est moins de 5; à 1/250 000, les tracés commencent à être dilatés, amplifiés, mais toujours semblables à la carte initiale; à partir de 1/500 000, il y beaucoup moins de catégories d'objets, de par la conception même de la carte, et par manque de place; à 1/1 000 000, le facteur de réduction atteint 40, la carte initiale occupe 10 mm^2, il n'en reste plus que la nationale N3, le chemin de fer, et la symbolisation ponctuelle de la ville.

Nous avons noté que la représentation graphique, moyen d'enregistement, de traitement et de communication de l'information géographique, est un langage qui a son vocabulaire et sa grammaire, et, par conséquent, ses règles d'orthographe et de syntaxe. Que, jusqu'à présent, ces règles aient été appliquées inconsciemment et sans formulation n'implique nullement leur inexistence et l'impossibilité de les énoncer de façon formelle.

En réalité, les propriétés des variables visuelles s'expriment sous forme de longueur, d'équivalence, d'ordre ou de rapports numériques. Les règles de lisibilité s'énoncent en termes numériques, les principes de généralisation, eux-mêmes, sont justiciables d'une formulation mathématique si on prend la peine de procéder à une approche logique et méthodique; loin d'affaiblir la portée de la généralisation, ces méthodes semblent la renforcer en lui apportant l'élément d'objectivité qui lui faisait défaut.

Le rôle personnel du cartographe reste fondamental dans un système automatisé conçu en mode *conversationnel*, c'est à dire permettant le dialogue permanent entre l'homme et la machine. Débarassé des travaux de routine et des contraintes artisanales, le cartographe de demain devrait être en mesure de mieux maîtriser les problèmes de conception tout en gardant des possibilités d'intervention, de choix et de décision au cours des phases d'exécution.

Conclusion générale

Dans cet exposé, on a abordé quelques aspects de l'automatisation de la généralisation cartographique dans le cadre des réseaux de communication. On a essayé de contourné les difficultés partiellement en faisant appel à la méthodologie des systèmes-experts.

On a montré essentiellement une analyse des objectifs (*pourquoi* généraliser), des estimations des situations (*quand* généraliser), une compréhension de problème (*comment* généraliser), et la recherche des méthodes informatiques en nous facilitant la vie pour réaliser une automatisation de ce problème bien complexe de la généralisation cartographique. Quelsque algorithmes basés sur la compréhension des éléments géographiques dans leur ensemble sont décrits.

On a ensuite présenté une approche globale de l'acquisition de la connaissance fondée sur un processus itératif en quatre phases: communication, formulation-modélisation, validation, réalisation-illustration. On a mis l'accent sur les études de l'acquisition des connaissances humaines, le savoir-faire des experts, leur modélisation mathématique, et également sur leur représntation informatique à l'aide de plusieurs outils pratiques de l'intelligence artificielle telles que: programmation orientée objet, la programmation par règles de production, la programmation en Lisp.

Il est difficile de dire telle technique informatique est meilleure que telle autre. Vue la complexité des problèmes de la généralisation et l'existence de nombreux de modes de représentation des connaissances, on est persuadé qu'aucun de ces modes ne constitue à lui seul une réponse au problème. En analysant les différents modes de représentation des connaissances tels que: les représentations procédurales, les représentations à base de règles, et les représentations par objets, on décrit une solution par l'utilisation conjointe de plusieurs techniques: représentation des connaissances de façon procédurale, par des objets, raisonnement avec des règles. Les connaissances abstraites de la GC seront ainsi codés de façon plus efficace et plus souple pour mieux traiter les problèmes complexes. A cette approche, nous avons associé diverses autres méthodes, pour résultat partiel que nous espérons intéressant. De ce point de vue, nous estimons, non pas avoir atteint un but d'un système totalement opérationnel, mais avoir pris la bonne direction, qui n'était pas évidente au début.

Je me rappelle qu'au début, je ne savait pas très bien où me mènerait cette reherche, et je n'avais pas non plus une idée très claire de l'objectif à atteindre. Car j'avais l'impression que, dans ce domaine, la communauté cartographe n'était pas optimiste. A l'issue de cette thèse, avec toutes les difficultés que j'ai eues, les résultats des études, des essais, j'ai maintenant une idée plus claire: l'automatisation de la généralisation n'est pas impossible, même tout à fait possible. Mais on ne peut pas s'attendre à des solutions parfaites à court terme. Elle nécessite des études au

niveau tant de la théorie cartographique que la pratique d'informatique. Il me semble qu'aucun de deux aspects ne constitue à lui seul une réponse au problème. La solution réside certainement dans le mariage de ces deux techniques bien différentes. L'informatique offre une grande facilité à la cartographie; la cartographie fournit un domaine particulier d'application qui demande l'évolution de l'informatique.

Les théories de la généralisation cartographique sont intéressantes à raconter, mais les difficultés résident dans la réalisation concrète de ce processus complexe dont la description est parfois un peu effrayante. C'est ce qui rend admirable le travail du cartographe, ou, de façon plus générale, de l'expert humain.

La résolution de ce problème résidera dans:

- la recherche des théories de la communication,
- une bonne représentation des connaissances et une formalisation des règles de généralisation utilisées inconsciemment par les experts humains.
- des algorithmes basés sur une compréhension de l'élément ou des éléments géographiques dans leur ensemble.

La recherche de la généralisation cartographique tend à approfondir les bases théoriques, à repenser les concepts de généralisation, d'algorithme, de modèles de données, et à développer des processus parallèles traitant des éléments cartographiques complexes.

Il semble qu'un système expert n'est pas seulement une base de règles et un moteur d'inférence, et que la construction de tels systèmes requiert plus que l'analyse de cohérence. C'est tout un environnement complexe qui doit être analysé pendant le développement d'un système expert et plus précisément pendant la phase d'acquisition de la connaissance qui ne se limite pas à une simple transmission d'informations.

Le système décrit dans cet exposé permet de modéliser un grand nombre de problèmes de la généralisation cartographique, grâce à la souplesse et à la puissance de repésentation de connaissances en objets, et au moteur d'inférences qui utilise des variables. Pourtant le cycle de base du moteur d'inférences est simple.

Le système peut également fonctionner en logique sans variable, mais il est moins efficace en rapidité qu'un système en pure logique de proposition. Le système est écrit en Le-lisp et Ceyx de l'INRIA. Un transfert vers Common lisp a été réalisé par monsieur Tierry Salset à l'IGN. Les éléments sommairement présentés dans cette communication sont en cours de perfectionnement.

Partie V

Annexe

Annexe A

Code des Programmes

A.1 Structuration et gestion interne des objets

Nous utilisons le même formalisme objet de façon homogène pour l'ensemble du programmme. Comme les éléments géographiques, les prédicats et les règles sont représentés par des objets, mieux adaptés pour l'ensemble des traitements. Ils ont donc des champs (variables d'instance ou de classe) et des méthodes (procédures) de traitement.

A.1.1 La représentation interne des faits

Un fait est une liste dont le premier élément est un prédicat suivi des arguments. Par exemple, dans `(Voisin a1 a2)`, `Voisin` est un prédicat, `a1` et `a2` ses deux arguments. Pour les arguments qui sont des variables, leurs valeurs sont souvent des objets, instances d'autres classes. Dans l'exemple précédent, `a1` et `a2` sont des routes ou des carrefours.

La définition de la classe {PREDICAT} de prédicats sous Ceyx par (Les explications des différents champs d'un prédicat se trouve dans la section 7.4.3):

```
(deftclass PREDICAT
           nom Lfaits-totale Lreg-ante Lreg-conc)
```

tandis que la fonction `make-Predicat` création de ses instances est:

```
(defmake {PREDICAT}
         make-Predicat
           (nom Lfaits-totale Lreg-ante Lreg-conc))
```

Fonction de traitement d'un fait

Traiter un fait consiste à actualiser les différents champs de son prédicat ou à en créer un dans un dictionnaire des prédicats (`dico`), les différents champs des règles qui contiennent ce prédicat. On suppose donc que son prédicat ne soit pas une variable:

```
(de traite-fait-rg1 ((pred . arg) . conc?);   le PREDICAT
 (or (VAR pred)  ; c'est une variable,
     (VARC pred) ; il n'y a rien a faire
     (selectq  pred
               ; aucune mise a jour a faire
      ((execute supprime liaison insere avec non echec))
       ; on passe l'action pour recommencer
       ; avec les arguments
      ((ajoute conclut)
               (traite-fait-rg1 (car arg) conc?))
      (t
        (unless
          (member pred dico)
                     ; le predicat n'est pas encore connu
                     ; du dictionnaire
          (set pred   ; on le cree, initialise
              (make-predicat pred '()
                             (pairlis '() '() ())
                             (pairlis '() '() ())))
              (newr dico  pred)) ; on le memorise
        (lets
          ((p-ou-c        ;conclusion ou antecedent
                (if conc? 'Lreg-conc
                          'Lreg-ante))
           (regles-quialefait
                (send p-ou-c
                     (eval pred)))) ; obtenir les regles
         (ifn
            (assq nom-regle  regles-quialefait)
         ; si la regle n'est pas presente dans le champ
            (and
                (send p-ou-c   ; alors on en rajoute.
                      (eval pred )
                      (acons nom-regle 1
                                     regles-quialefait))
         ; si on traite une premisse, on incremente le
                (or conc? ;nombre de predicats de regle
                (sendq nb-predicats
                       regle
                       (incr nb-pred))))
        ; sinon la regle a deja une instance de ce predicat
        ; on increment le nb d'instance
                (send p-ou-c
                      (eval pred )
                      (al-vincr nom-regle
                      regles-quialefait))))))))
```

A.1.2 La représentation interne des règles

La classe de règles {REGLE} est définie ci-dessous:

```
(deftclass REGLE
            nom
            utilisee?
            coefficient
            variables
            nb-predicats
            nb-predicats-inconnu
            antecedents
            consequents)
```

la création des instances de règles par la fonction *make-regle*:

```
(defmake {REGLE}
          make-regle
              (nom
               utilisee?
               coefficient
               variables
               nb-predicats-inconnu
               nb-predicats
               antecedents
               consequents))
```

Par exemple, la regle R3.1 est une instance de la classe {REGLE}, elle aura la forme interne:

```
(setq
   R3.1
   (make-regle    ;la fonction d'appel de creation d'instance
         'R3.1                        ;le nom
         '()                          ;non-utilisee
         '1                           ;coefficient
         '(?obj1 ?obj2 ?obj3)         ;les variables
         '2                  ;le nombre de predicats inconnu
         '((Ecarter . 2))             ;nombre de predicat
         '((Ecarter ?obj1 ?obj2)      ;   antecedent1
           (Ecarter ?obj1 ?obj3)      ;   antecedent2
           (avec (nequal ?obj2 ?obj3)))    ;   antecedent3
         '((ajoute (Ecarter-g ?obj1 (?obj2 ?obj3))); conclusion1
           (supprime (Ecarter ?obj1 ?obj2))        ; conclusion2
           (supprime (Ecarter ?obj1 ?obj3)))))     ; conclusion3
```

Les macros de reconnaissance lexicale: VAR, VARC, VARCN

Ce sont des définitions des variables pour les prédicats et le filtrage.

Déterminer des variables simples consiste de voir si sa première lettre est commencée par ?: variable::= **?<identificateur>**. C'est une variable filtrant un objet:

```
(dmd VAR (element)
     '(and (variablep ,element)
           (= (chrnth 0 ,element) #/?)))
```

Déterminer les variables segments::= **&<identificateur>**. C'est une variable filtrant une liste

```
(dmd VARC (element)
     '(and (variablep ,element)
           (= (chrnth 0 ,element) #/&)))

? (VARC '&x+)
= 38
```

Une variable segment non anonyme (i.e ne commençant pas par **&-**):

```
(dmd VARCN (element)
     '(and (VARC ,element)
           (neq (chrnth 1 ,element) #/-)))
```

Initialisation

Initialise les différentes variables qui se trouvent dans des fonctions différentes:

```
(de initvar ()
       (setq mode 't
               ; imprimer les messages de deduction premier degre
             mode-infere '()
               ; imprimer les messages de deduction 2ieme degre
             mode-aff-reg '()
               ; imprimer les messages d'affiche des regles
             env '((i i))  ; environnement
             cpt 0
             base_de_faits ()
             base_de_regles ()
             dico ()  ; dictionnaire des predicats
             definitions ()
             implicites ())) ; les faits initiaux
```

Fonctions de traitement des variables

L'**env**: une **A-list**, est une variable globale décrivant un environnement pour le filtrage et l'unification.

Livrer toutes les instances d'un predicat:

```
(de fait-inst (predicat)
    (sendq Lfaits-inst predicat))
```

Retourne la valeur terminale d'une variable dont la valeur peut être une variable:

```
(de Valeur (var)
     (let ((v (cassq var env)))
          (if v          ; si la valeur est une variable,
             (Valeur v)  ; on l'evalue recursivement
              var)))     ; sinon on rend le resultat

 ? env
 = ((&x+ . s) (&x . i) (&x . s) (&x . s) (i i))
 ? (Valeur &x)
 = s
```

Ajout d'un fait

On ajoute un fait dans la base de faits (`base_de_faits` est une variable globale) soit par initialisation de la base de faits, soit par des déductions, des déclenchement de règles par le moteur d'inférence. Il faut donc prendre en compte des effectives d'un nouveau fait dans la base pour actualiser l'état des règles, etc. On s'interdit d'introduire des faits déjà connus.

```
(de ajoute (fait . top?)
         ; le test d'existence est necessaire lors d'une
         ; deduction par le systeme.
   (unless
     (member fait base_de_faits)
         ; si ce fait non encore connu
         ;1- affichage de la deduction
         ; suivant le mode et le moment de linsertion.
     (when mode ; on a le droit de parler
          (prin "---- Ajout de :       " fait)
                ; si l'insertion a lieu au Toplevel aucune
                ; regle n'est declenchee
          (if top?
              (terpri)
              (print "  avec la " nom-regle)))
               ;2-on actualise le dictionnaire de predicats
     (dico-newp (car fait))
               ;3-on actualise la base de faits
               ; on l'insere dans la base de faits
               ;4-on actualise le champ Lfaits-inst de ce
               ;predicat et ajoute cette nouvelle instance
               ;  sous le champ Lfaits-ins
     (sendq Lfaits-inst
```

```
    (eval (car fait))
    (cons fait
    (sendq Lfaits-inst (eval (car fait)))))
     ; 5-on actualise les champs Lreg-ante,
     ; Lreg-cons du predicat
     ; et chaque regle ou le predicat est premisse
(mapc
  (lambda ((regle . nb-instance))
    (if
      (Pred-nega regle)  ; une regle avec
              ;  premisse negative?
      (when
        (= 1
           (sendq nbfconnu (eval (car fait))))
                  ;premier fait ajoute
         (lets ((reg (concat (substring regle 1)))
                    ; premisse negative
           (nb-pr-in (sendq nb-pred-inco
                            (eval reg )))
           (nb-pr    (sendq nb-predicats
                            (eval reg ))))
         (unless
          (sendq utilisee
                (eval reg)); sinon utilisee
          (sendq nb-pred-inco
                 (eval reg)
                 (incr nb-pr-in)))))
    ; premisse positive
      (let ((nb-pr-in (sendq nb-pred-inco
                             (eval regle )))
            (nb-pr    (sendq nb-predicats
                             (eval regle ))))
       (ifn
         (sendq utilisee   (eval regle))
      ;*1 si elle n'a pas ete declenchee et qu'on
      ; insere une nouvelle instance du predicat
         (or
          (eq 0  nb-pr-in)
             ;soit plus de premisse inconnue
             ;pas la peine de continuer
          (< nb-instance     ; soit le besoin de
            (sendq nbfconnu ; ce fait de la regl
                   (eval (car fait))))
              ; est moins grand que ce qu'on a
              ; sinon ajout de ce nouveau
          (sendq nb-pred-inco   ; fait decr
                 (eval regle)    ;le nb-pred-inco
                 (decr nb-pr-in))) ; de cette regle
```

```
          ;*2 la regle a deja ete declenchee, on la rend
          ;  donc candidate remettant son compteur a 0,
          ;  on la marque comme declenchable
               (sendq nb-pred-inco
                      (eval regle) 0)
               (sendq utilisee
                      (eval regle) '())))))
                ; ens de regles ou le predicat est premisse
     (sendq Lreg-ante  (eval (car fait))))
                ; pour envoyer un resultat T pour succes
     (newr base_de_faits fait)))
```

Suppression d'un fait

La suppression d'un fait de la base de faits caractérise la non monotonie du système.

```
; pour supprimer des faits etablis(non-monotonie)
(de supprime-fait (fait)  ; supprime un fait ou un filtre
    ; on ne supprime pas un atom, un fait est une liste
   (if (atom fait)
       (error 'suppression "non autorisee" fait)
       (enleve (filtrep fait) t)))
```

Prise en compte effective de la suppression.

```
(de enleve (faits . top?)
  (mapc
   (lambda (fait)
     (when        ;si le fait n'est pas connu, on
       (member fait base_de_faits) ; ne fait rien
                ; si le mode d'affichage le permet,
       (when mode ;  on imprime chaque retrait
            (prin "----SUPPRESSION de: " fait)
                ; au toplevel aucune regle n'est appliquee
            (if top?
               (terpri)
               (print " avec la " nom-regle)))
                ; on retire le fait de la base de faits
       (setq base_de_faits
             (retire fait base_de_faits))
         ; on retire le fait des instances connues du predicat
       (sendq Lfaits-inst
             (eval (car fait))
             (retire fait
                    (sendq Lfaits-inst
                           (eval (car fait)))))
         ; s'il n'y a plus d'instance du predicat
```

```
(let ((nbf (sendq nbfconnu (eval (car fait)))))
     ;pour les regles ou le predicat est premisse
 (mapc
  (lambda ((regle . nb-instance))
    (if (Pred-nega regle)
        (unless
          (= nbf 0)
          (lets ((reg (concat (substring regle 1)))
                                ; premisse negative
                 (nb-pr-in
                     (sendq nb-pred-inco
                            (eval reg)))
                 (nb-pr
                     (sendq nb-predicats
                            (eval  reg ))))
            (or (= 0 nb-pr-in)
               (sendq nb-pred-inco  (eval  reg)
                                    (decr nb-pr-in)))
               (when (sendq utilisee (eval  reg))
                     (sendq utilisee (eval reg) '()))))
        (when       ; premisse positive
          (< nbf nb-instance)
                    ; moins de fait que ce qu'il faut
          (let ((nb-pr (sendq nb-pred-inco
                              (eval regle))))
            (ifn  ; si la regle n'a pas ete utilisee ou
             (sendq utilisee (eval regle))
                    ;rendue declenchable
                    ; alors on incremente son compteur
             (sendq nb-pred-inco (eval regle)
                                 (incr nb-pr))
                  ; sinon on la rend non-utilisee
             (sendq utilisee (eval regle) '()))))));
                          ;toutes les regles
      (sendq Lreg-ante (eval (car fait)))))))
faits)) ;pour tous les faits
```

A.2 Le filtrage

Le filtrage est employé aussi bien pour les inférences déductives que pour la gestion interne des connaissances. Il permet de sélectionner l'ensemble des faits par un filtre. Le déclenchement des règles est réalisé en filtrant les prémisses des règles avec des éléments de la base de faits.

Evaluer une expression exécutable (le CAR est une fonction exécutable) dans un environnement représentée par une A-liste telle que `((?x p1) (?y p2))`.

```
(de evalexp(exp env)
 (if (constantp  exp) ;une constante
```

```
    exp              ; on la rend comme r\'esultat.
    (if  env ;on construit une fermeture
             ;dans laquelle on evalue l'exp
         (apply '(lambda ,(mapcar 'car env) ,@exp)
                 (mapcar 'cdr env))
         env)));sinon on rend l'environnement vide
```

Substituer une liste de variables (qui existent déjà dans l'**env**) dans une expression, les variables non figurées dans **env** ne sont pas remplacées

```
(de Substr (exp env)
  (if env
      ; appel recursifs tant que tout
      ; l'environnement n'a pas ete visite
     (Substr
       (if (VARC (caar env))
             ; variable segment, le resultat etant
             ; une liste, on fait appel a la fonction "MAPCAN"
           (Substs (cdar env) (caar env) exp)
             ;sinon on substitue avec "SUBST"
           (subst (cdar env) (caar env) exp))
       (cdr env))
     exp)); on a parcouru tout l'environnement,
              ; on rend la nouvelle exp.
```

Pour substituer une variable segment dans une expression:

```
(de Substs (valeur variable exp)
  (mapcan (lambda (element)
            (if (equal variable element) ; c'est une variable
                valeur ; la valeur est une liste
                       ; sinon, on cree une liste
                (list element))) ;(pour le MAPCAN)
     exp))
```

Dans la fonction suivante: l'argument **filtre** contient des variables (**?x**, **?-**, **&x**, **&-**), supposons que le **CAR** de (**filtre**) soit un prédicat non variable, la fonction rend comme résultat la liste des faits filtrables avec ce **filtre** dans le champ **Lfaits-inst** de son prédicat dans l'**env**(ironnement).

```
(de Ffiltres (filtre env)
       ; on memorise tous les faits connus
       ; du predicat du filtre
      (let ((faits (sendq Lfaits-inst
                          (eval (car filtre)))))
             ; ensuite on les filtre
           (Filtre-donnees faits filtre env)))

? env
```

```
  = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (setq Deplacer
      (make-predicat
       'Deplacer
       '((Deplacer p1 p2)
         (Deplacer p2 p3))
       '(r1 r2)
       '(r2)))
   = #(#:Tclass:Predicat . #[Deplaccer
        ((Deplacer p1 p2)
         (Deplacer p2 p3))
         (r1 r2)
         (r2))
? (Ffiltres '(Deplacer ?- ?-) env)
   = ((Deplacer p1 p2) (Deplacer p2 p3))
;; les deux faits sont filtrables
? (Ffiltres '(Deplacer ?p1 ?-) env)
   = ((Deplacer p1 p2))
;; un seul, puisque ?p1 vaut p1 dans env.
```

Comparer un **filtre** avec une liste des données **ldonnees**: rendre l'ensemble des faits filtrés.

```
(de Filtre-donnees (ldonnees filtre env)
  (and ldonnees ; s'il y des donnees, on continue le filtrage
      (if (Filtre (car ldonnees) filtre env)
                  ;si le car est filtrable,
                  ;on la memorise dans une liste
          (cons (car ldonnees)
                (Filtre-donnees (cdr ldonnees)
                                filtre
                                env))
                ; sinon, on regarde le reste: cdr
          (Filtre-donnees (cdr ldonnees) filtre env))))

? env
  = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (Filtre-donnees '((poser p1 p2) (poser p1 p3))
                  '(poser ?p1 ?p2) env)
  = ((poser p1 p2))
? (Filtre-donnees '((poser p1 p2) (poser p1 p3))
                   '(poser ?p1 ?-) env)
  = ((poser p1 p2) (poser p1 p3))

? (Filtre-donnees '((poser p1 p2) (poser p1 p3))
                  '(poser &-) env)
  = ((poser p1 p2) (poser p1 p3))
```

Le filtrage **Filtre** compare un **filtre** qui contient des variables avec une **donnee**, il rend l'environnement éventuellement augmenté si le filtrage est réussi.

```
(de Filtre (donnee filtre env)
  (and env; non nul
       (cond
         ((VAR filtre); c'est une variable simple
               (Filtres donnee filtre env))
         ((atom filtre)
                ; un atom: les 2 doivent etre identiques
               (and (eq filtre donnee)
                    env))
         (t  ; sinon le filtre est une liste
             (Filtre-liste donnee filtre env)))))


? env
  = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (Filtre '(deplacer p1 p2) '(&-) env)
  = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (Filtre '(deplacer p1 p2) '(&x) env)
   = ((&x deplacer p1 p2) (?p1 . p1) (?p2 . p2) (?p3 . p2)
      (?p4 . ?p3))
? (cadr (Filtre '(deplacer p1 p2) '(&x) env))
  = (?p1 . p1)
```

Pour comparer un **filtre** qui est une liste avec une donnee.

```
(de Filtre-liste (donnee filtre env)
  (and env
      (cond
        ((null filtre);si le filtre est vide, teste de fin
                    ; la donnee doit l'etre aussi
              (and (null donnee)
                   env)) ; rend env comme resultat

               ; son CAR est une variable segment complexe
        ((VARC (car filtre))
              (Filtrec donnee filtre env))

              ; le CAR du filtre annonce une condition a verifier
        ((eq (car filtre) '/)
                    ; on les evalue
          (and (evalexp (list (cadr filtre)) env)
                    ;on continu  le filtrage
                    ;si la condition est verifiee
               (Filtre-liste donnee (cddr filtre) env)))

         (t   ; on a une liste non vide
           (and (consp donnee)
                (Filtre-liste   ; on filtre  les CDR
                    (cdr donnee)
```

```
            (cdr filtre)
          ; dans env eventuellement enriche avec le CAR
            (Filtre (car donnee)
                    (car filtre)
                    env)))))))
```

La fonction **Filtres** compare une variable simple de type (?x ou ?-) avec une donnee dans l'environnement: **env**.

```
(de Filtres (donnee variable env)
  (if (eq variable '?-)
      env   ; il n'y a pas de liaison, on rend env inchange
     (let  ((valeur (cassq variable env)))
      (if valeur ; si la variable est deja presente dans env
           ;on teste l'egalite sa valeur avec la donnee. si
           ; identiques, on rend env inchange, sinon echec
          (and (equal donnee valeur)
               env)
        ; si elle non encore presente dans env
        ;  on lui affecte la donnee,  env est enrichit
          (newl env (cons variable donnee))))))
```

La fonction **Filtrec** compare un filtre dont le premier est une variable segment de type (**&x** ou **&-**) avec une donnee dans l'environnement: **env**.

```
(de Filtrec (donnee filtre env)
  (let  ((valeur  (assq (car filtre) env)))
    (cond
      (valeur
         ; il y a une valeur, on substitue la variable par
         ; sa valeur dans le filtre, on recommence le filtrage
         ; avec ce nouveau filtre et la meme donnee
      (Filtre-liste donnee
                    (append (cdr valeur)
                            (cdr filtre))
                        env))
     ((null (cdr filtre))
         ; il n'y a plus rien dans le filtre apres la variable.
         ; si cette variable est anonyme(&-), on rend env,
         ; sinon on cree une nouvelle liaison variable-valeur
         ; avec toute la liste "donnee" en valeur, rend env
       (and (listp donnee)
            (Res (cons (car filtre) donnee))))

      (t ; sinon on initialise la variable avec
          ;un segment de list vide.
          (let  ((liaison (list (car filtre)))
                 (donnee donnee))
          (Filtrec1 liaison donnee))))))); essaye la liaison
```

Filtrec1 essaye de faire adapter les liaisons pour la variable &- ou &x avec une donnnee.

```
(de Filtrec1(liaison donnee)
  (and donnee
            ;si la donnee devient nulle, on reste
            ; sur echec, et on rend nil.
            ;si le filtrage reussit, la liaison de notre
            ; variable convient, on sort du OR  avec env
       (or (Filtre-liste donnee
                         (cdr filtre)
                         (Res liaison))
            ;il y a un echec avec la liaison, on essaye
            ;donc avec un segment de liste superieur
           (Filtrec1
                (append liaison  ; a cause de NEXTL
                        (list (nextl donnee)))
            ;on est dans CDR de donnee du pas precedent
                  donnee))))

? env1
= ((?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (Filtre '(deplacer p1 p2 p3) '(?d &x ?w) env1)
= ((?w . p3) (&x p1 p2) (?d . deplacer) (?p2 . p2)
   (?p3 . p2) (?p4 . ?p3))

? (Filtre '(deplacer p1 p2 p3) '(?d &- ?w) env1)
 = ((?w . p3)(?d . deplacer)(?p2 . p2)(?p3 . p2)(?p4 . ?p3))
; le &- a filtre (p1 p2)

? (Filtre '(deplacer p1 p2 p3) '(?d &- ?w ?e) env1)
 = ((?e . p3) (?w . p2) (?d . deplacer) (?p2 . p2)
    (?p3 . p2) (?p4 . ?p3))
; le &- a filtre (p1)

? (Filtre '(deplacer p1 p2 p3) '(?d &x  ?w ?e) env1)
 = ((?e . p3) (?w . p2) (&x p1) (?d . deplacer) (?p2 . p2)
    (?p3 . p2) (?p4 . p3))
; le &x a capture (p1)
```

Res rend **env** inchangé si on a affaire à une variable anonyme, ou enrichit **env** si la variable est nommée.

```
(de Res (liste)
      (if (= (chrnth 1 (car liste))  #/-)
          env
          (cons liste env)))
```

La fonction **filtrep** donne la liste des faits a enlever lors d'une action **supprime**.

On ne fera appel à la fonctions **Ffiltres** que si l'on a un filtre qui contient une variable anonyme (**?-** ou **&-**); dans ce cas on aura plusieurs éléments à supprimer.

```
(de filtrep (donnee)
    (if (or (occur '|&-| donnee)
            (occur '|?-| donnee))
        (Ffiltres donnee env) ;si on a une variable anonyme
        (list donnee))) ; sinon on rend la donnee elle-meme
```

Livre la liste de tous les faits filtrables connus lors d'une question.

```
(de interroge (donnee)
     (let ((faits (Ffiltres donnee env)))
                        ; les faits filtrables
          (ifn faits
               (print " Il n'y a pas de fait")
               (mapc 'print faits) ; il y en a , les afficher
               (print "Et voila , c'est tout"))))
```

A.3 L'unification

Le filtrage compare un filtre pouvant contenir des variables et une donnée sans variable. L'unification, quant à elle autorise la comparaison de deux données pouvant contenir des variables. Le filtrage est donc un cas particulier de l'unification: on parle aussi de semi-unification.

Pour notre unification, les variables autorisées ne sont que d'un seul type: **VAR** qui sont des variables simples nommées (ex: **?x, ?objet**).

La fonction **unifie** rend **env** créé si l'unification a réussi, **nil** dans le cas contraire.

```
(de unifie (e1 e2 env)
 (and env      ; si l'env est nul, on arrete tout
     (cond     ; les terms sont egaux, rend l'env
      ((eq e1 e2)
        env)      ; pour les termes liste identiques,
                  ; seront verifies ici
      ((VAR e1)   ; on ne peut substituer une variable par
                  ; une expression la contenant
       (if (occur e1 e2)
           ()     ;echec
                     ;sinon, on enrichit l'environnement
           (if          ;si e1 n'est pas dans env
              (member (cons e1 e2) env)
              env
              (newl env (cons e1 e2)))))
      ((VAR e2)     ; comme precedemment
       (if (occur e2 e1)
           ()   ;echec
           (if          ;si e1 n'est pas dans env
```

```
            (member (cons e2 e1) env)
            env
            (newl env (cons e2 e1)))))
      ((or (atom e1 ) (atom e2))
       ()); 2 atoms differents, echec
            ; diviser pour mieux regner:
            ; on unifie dans les CAR et dans les CDR
      (t (unif-suite (cdr e1)
                     (cdr e2)
            ;avec env enrichi dans les CAR
         (unifie (car e1)
                 (car e2)
                 env))))))
? env1
 = ((?p2 . p2) (?p3 . p2) (?p4 . ?p3))

? (unifie1 '?p1 '?p1 env1)
 = ((?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (unifie1 '?p1 'p1 env1)
 = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
```

Les differences entre **unifie** et **unifie1**

```
? env
 = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3))
? (unifie1 '?p1 'p1 env)
 = ((?p1 . p1) (?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3)
   (&x p1 p2))
? (unifie1 '(?p1 ?pq p2 p1) '(p1 p1 ?pe p1) env)
 = ((?pe . p2) (?pq . p1) (?p1 . p1) (?p1 . p1) (?p2 . p2)
     (?p3 . p2) (?p4 . ?p3) (&x p1 p2))
```

Dans le **unifie1**, il n'y a pas de détection d'existence de valeur d'une variable déjà présente dans l'environnement, donc il y a la repétition de (?p1 . p1) dans env.

Alors que la fonction **unifie** n'aura pas ce défaut.

```
? (unifie '(?p1 ?pq p2 p1) '(p1 p1 ?pe p1) env)
 = ((?pe . p2) (?pq . p1) (?p1 . p1) (?p2 . p2) (?p3 . p2)
    (?p4 . ?p3) (&x p1 p2)) pas de  repetition de (?p1 . p1)

? (unifie '(?p1 ?pq p2 p1) '(p1 p1 ?pe &-) env)
     ;  &- doit unifier une liste
  = ()
? (unifie '(?p1 ?pq p2 p1) '(p1 p1 ?pe &x) env)
      ; de meme &x
  = ()
? (unifie '(?p1 ?pq p2 p1) '(p1 p1 ?pe ?var.simple) env)
  = ((?var.simple . p1) (?pe . p2) (?pq . p1) (?p1 . p1)
     (?p2 . p2) (?p3 . p2) (p4 . ?p3) (&x p1 p2))
```

unifie1 ne fait pas de vérification de duplication dans env:

```
(de unifie1(e1 e2 env)
  (and env           ; si l'env est nul, on arrete tout
      (cond
                     ; les terms sont egaux, rend l'env
       ((eq e1 e2)
        env)  ; pour les termes liste identiques,
                     ; seront verifies ici
       ((VAR e1)
                     ; on ne peut substituer une variable par
                     ; une expression la contenant
        (or (occur e1 e2)
                    ;sinon, on enrichit l'environnement
            (newl env (cons e1 e2))))
       ((VAR e2)
              ; comme precedemment
        (or (occur e2 e1)
            (newl env (cons e2 e1))))
       ((or (atom e1 ) (atom e2))
        ()); 2 atoms differents, echec
           ; diviser pour mieux regner:
           ; on unifie dans les CAR et dans les CDR
       (t (unif-suite (cdr e1)
                      (cdr e2)
                        ;avec env enrichi dans les CAR
                      (unifie1 (car e1)
                               (car e2)
                                env))))))
```

On substitue une liste des variables déjà présentes dans env par leur valeur dans e1 et e2.

```
(de unif-suite (e1 e2 env)
       (unifie (Substr e1 env) ;substituer une liste de
                               ;variables dans une expression
               (Substr e2 env)
               env))
```

La fonction occur teste l'occurrence d'un élément dans une expression liste.

```
(de occur (element liste)
      (if (atom liste)
                ; on teste l'egalite avec un atome
          (eq element liste)
                ; si l'element apparait dans le CAR,
                ; on ne teste pas dans le CDR
          (or (occur element (car liste))
              (occur element (cdr liste)))))
```

A.4 Code du moteur d'inférence

Substitution des variables de la conclusion d'une règle par leur valeur trouvée dans l'environnement lors du déclenchement.

```
(de genere (exp)
 (mapcan
  (lambda (terme)
   (if (VARCN terme) ;; variable segment non anonyme,
                     ;  sa valeur sera forcement une liste
       (copy (Valeur terme))
             ; dans les autres cas, on cree une liste(MAPCAN)
       (list
         (cond
             ; c'est une variable;COPY important!!!
          ((VAR terme) (copy (Valeur terme)))
                       ; c'est une constante
          ((atom terme)  terme)
                     ; une liste, et c'est une fonction Lisp
          (t (if (typefn (car terme))
                     ; on evalue
                 (evalexp (list terme) env)
                     ; liste de liste, continuer
                 (genere terme)))))))
   exp))
```

Une variante de la fonction précédente, substitue les variables sans évaluer les fonctions.

```
(de genere-neval (exp)
 (mapcan
  (lambda (terme)
    (if (VARCN terme) ;; variable segment non anonyme,
                      ;  sa valeur sera forcement une liste
        (copy (Valeur terme))
          ; dans les autres cas, on cree une liste(MAPCAN)
        (list
         (cond
               ; c'est une variable;COPY important!!!
          ((VAR terme) (copy (Valeur terme)))
               ; c'est une constante
          ((atom terme)  terme)
               ; une liste, et c'est une fonction Lisp
          (t (genere-neval terme))))))
               ;liste de liste, continuer
    exp))
? env
 = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (&x p1 p2))
? (genere '(?p1 ?p2 ?p3))
```

```
 = (p1 p2 p2)
? (genere '(proche ?p1 ?p2))
 = (proche p1 p2)
? (genere '((proche ?p1 ?p2) (deplacer ?p1 p2)))
 = ((proche p1 p2) (deplacer p1 p2)
? (genere '(pd p))
 = (pd p)
? (genere '((proche ?p1 ?p2) (deplacer ?p1 &x)))
 = ((proche p1 p2) (deplacer p1 p1 p2))
```

; pour enrichir l'ENV d'une liaison au cours du chainage avant

```
(de Enrichirav (liste)
   (newl env
        (cons (nextl liste); on place la nouvelle variable
                  ; on place sa valeur (exp evalue)
        (evalexp liste env))))

? env
 = ((?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3) (&x p1 p2))
? (Enrichirav '(?pp ?p1)))
 = ((?pp . p1) (?p1 . p1) (?p2 . p2) (?p3 . p2) (?p4 . ?p3)
    (&x p1 p2))
? (Enrichirav '(?pp 'p1))
 = ((?pp . p1) (?pp . p1) (?p1 . p1) (?p2 . p2) (?p3 . p2)
    (?p4 . ?p3) (&x p1  p2))
? env
 = ((?pp . p1) (?pp . p1) (?p1 . p1) (?p2 . p2) (?p3 . p2)
    (?p4 . ?p3) (&x p1 p2))

? env
 = ((i i))
? (Enrichirav '(?x 'x))
 = ((?x . x) (i i))
? (Enrichirav '(?y 'y))
 = ((?y . y) (?x . x) (i i))
? (Enrichirav '(&xy (cons ?x ?y)))
 = ((&xy x . y) (?y . y) (?x . x) (i i))
? (Enrichirav '(&xly (list ?x ?y)))
 = ((&xly x y) (&xy x . y) (?y . y) (?x . x) (i i))
? (Enrichirav '(&xay (append ?x ?y)))
 =((&xay . y) (&xly x y) (&xy x . y) (?y . y) (?x . x) (i i))
```

Insérer une liste d'éléments **elements** dans une **liste**:

```
(de ajoutdR (elements liste)
    (unless (member (car elements)
                    liste)
```

```
            (nconc liste
                   (list (car elements)))
            (when (cdr elements)
                  (ajoutdR (cdr elements)
                           liste))))
```

Insérer un élément atomique:

```
(de ajoutdR2 (element liste)
       (unless (member element liste)
               (nconc liste
                      (list element))))
```

Retirer un élément: retire une seule occurrence d'un élément dans une liste:

```
(de retire (element liste)
    (ifn (member element liste) ; tester l'existence
         liste
        (if (equal (car liste) element)
            (cdr liste)
            (rplacd liste
                    (retire element
                            (cdr liste))))))
```

Moteur d'inférence

La classe de moteur d'inférence:

```
(deftclass Moteur-d-inferences
                              base-de-regles
                              base-de-faits
                              bdff
                              regles-valides)
(defmake {Moteur-d-inferences}
                         MI
                            (base-de-regles
                             base-de-faits
                             bdff
                            regles-valides))

(defvar mi1 (MI  () () () ()))
```

Initialise le moteur

```
(de {Moteur-d-inferences}:initialise-mi (mi)
            (setq mi (MI () () () ()))
            (reinitialise)
            (sendq regles-valides mi '())
            (sendq base-de-faits mi base_de_faits)
```

```
          (sendq base-de-regles mi base_de_regles)
          (sendq Valider-r mi)
       mi)
```

Afficher la règle:

```
(de {Moteur-d-inferences}:afficher (mi)
     (mapc (lambda (regle)
             (sendq afficher (eval regle)))
           (sendq base-de-regles mi)))
```

Trouver l'ensemble des règles valides:

```
(de {Moteur-d-inferences}:Valider-r (mi)
    (sendq regles-valides mi ())
    (mapc (lambda(regle)
            (when (and (sendq valide? (eval regle))
                       (not (member regle
                                    (sendq regles-valides
                                           mi))))
                  (sendq regles-valides mi ;
                         (nconc (sendq regles-valides mi)
                                (list regle)))))
          base_de_regles)
    (sendq regles-valides mi))
```

Le chaînage avant réalise des déductions, il envisage les règles candidates dans l'ordre ou elles ont été fournies, de même pour les prémisses. Une règle est déclenchée lorsque toutes ses prémisses ont été instanciées par des faits . Les actions de la règle sont alors envisagées dans l'environnement créé lors des instanciations des prémisses.

```
(de {Moteur-d-inferences}:en-avant (mi)
    (sendq initialise-mi mi)
    (while  ; on sature sur la base de regles
         (new-cyc mi))
       (let ((mode '()) )
                ; enlever les faits initiaux
       (enleve implicites t)
                ; afficher les faits d\'eduits
       (print " = :" base_de_faits)
       (terpri)))
```

Dans un nouveau cycle:

```
(de new-cyc (mi)
   (let ((regles-v (sendq Valider-r mi) ))
      (and regles-v ; il n'y a plus de regles candidates,
                    ; on a sature la base de connaissance.
                    ; une regle s'est appliquee avec succes
          (if (sendq essai-regle (eval  (nextl regles-v)) 0)
```

```
                ; on recommence avec la nouvelle regles-v
            (new-cyc mi)
            (same-cyc regles-v)))))
```

Dans le même cycle:

```
(de same-cyc (bdr-v)  ;balaye la bdr
    (ifn bdr-v
                ; Plus de regle valable. FIN de Cycle
        (if  (sendq essai-regle
                  (eval (nextl bdr-v)) 0)
             (new-cyc mi)
             (same-cyc bdr-v))))
                ; sinon on passe a la suivante
```

Détection : déclencher une régle, l'ensemble de régles candidates est virtuellement construit par l'état de la base de faits.

```
(de {Regle}:essai-regle (regle nb-succes)
  (when
    (sendq valide? regle)
    (let ((nom-regle (sendq nom regle)))
      (sendq afficher regle )
            ; on essai de la saturer avec la base de faits
      (infere (sendq antecedents regle) env)
      (sendq utilisee regle 't);marque comme utilisee
            ;(elle n'est plus potentiellement utilisable)
      (sendq nb-pred-inco regle 1)
      (ifn (> nb-succes 0)
           (print "-on n'a pas eu de succes avec la regle- : "
                  nom-regle)
           (print  nb-succes
                  " succes avec la regle- : "
                  nom-regle "
                  <<FIN>> de la saturer!"))
      (and mode-infere
          (print  "-la base_de_faits actuel est-- : " )
          (print "******"
                  base_de_faits))
         (terpri)
      (> nb-succes 0))))
```

Déclencher une règle d'exécution de fin:

```
(de {Regle}:exeaction (regle)
      (when (sendq valide? regle)
            (infere (sendq antecedents regle)
                    env)))
```

Inférence:

```
(de infere (antes env)
 (and env   ; si l'environnement est nul, il y a echec
     (ifn   ; plus de premisses,
      antes ;on envisage la partie action de la regle
     (succes (sendq conclusions regle) env)
           ;sinon on commence a inferer
     (selectq (caar antes)
           ; acces au predicat ou operateur de la premiere
           ; premisse une contrainte lisp  evalue-la
      ((avec execute)
       (and
         (evalexp (cdar antes) env) ; on continue
           ; si l'evaluation est differente de NIL
         (infere (cdr antes) env)))
            ; une liaison, enrichir l'env
      (liaison
          (Enrichirav (cadar antes) )
          (infere (cdr antes) env))
        ; operateur de non demontrabilite (monde clos)
     (echec
       (or (prouver  ; pas encore defini
                (Substr (cadar antes) env)
                env)
               ; si la resolutiona echoue on continue
                ; sinon Retour-arriere
           (infere (cdr antes) env)))
       ; la negation (monde clos)
     (non       ;si aucune des instances du predicat considere
                ; ne filtre la condition
       (or
        (any
          (lambda (fait)
           (Filtre fait
                  (cadar antes)
                   env))
           (sendq         ;ens des instance du predicat
              Lfaits-inst
             (eval (Valeur (caadar antes)))))
           ;si le fait ne parait pas dans la bdf, on continue
         (infere (cdr antes) env)))
      ; c'est une premisse simple
     (t
      (mapc
          (lambda (fait) ; on sature sur la regle
           (infere (cdr antes) ; on instancie une premisse
                   (Filtre fait
                          (car antes)
                           env)))
```

```
        (sendq Lfaits-inst
               (eval  (Valeur (caar antes))))))))))
```

Execution de la partie action d'une règle

```
(de succes (liste-exp env)
 (tag ajoute
       ; on s'echappe si une conclusion est deja connue
   (mapc
    (lambda (conc)
       ; si l'action est une "execute", on ne substitue pas les
       ; variables par leur valeurs, la fonction d'evaluation
       ; "evalexp" realisant cette substitution
    (let
      ((conclusion
         (if (eq (cadr conc)
                 'execute)
             (cdr conc) ; on ne fait rien
                        ; on substitue
             (genere (cadr conc)))))
      (selectq (car conc)
       (execution     ; action
          (evalexp (cdr conc) env))  ;evaluation LISP
       (execute       ; c'est une execution
          (evalexp conclusion env)) ; on l'evalue
               ; insertion d'un fait dans la base de faits
       ((ajoute insere)  ;un ajout de conclusion
          (or (ajoute conclusion)
                      ;si la conclusion est deja connue
              (and (print " --- ajout non reussi! ")
                   (exit ajoute)))
          (incr nb-succes))    ;ajout reussi !!
       (supprime ; suppression de faits
                 ; la fonction "filtrep" rend l'ens de tous les
                 ; faits filtrables par la conclusion
         (enleve (filtrep conclusion)))
       (conclut ) ;une simple conclusion
                 ; rien a faire,(pour chainage arriere)
                 ; l'action envisagee n'est pas connue
       (t (error 'succes
                 "selecteur inconnu"
                 (car conc))))))
  liste-exp)))
```

Annexe B

Art de PostScript

Il me semble que je ne peux pas, dans cette communication, passer sous silence le langage PostScript (PS en abbreviation ci-dessous), sans lequel je n'aurais pas pu illustrer les travaux de cette thèse. On a cherché depuis le début un logiciel graphique adéquate en vain, mais ce n'est pas la seule pour laquelle ce langage nous a attiré, ni le succès que PostScript a déjà eu en milieu industriel. La raison principale est peut-être sa puissance et sa simplicité.

B.1 Introduction

PostScript est un langage de description de page, fruit de plusieurs années de recherche. Le premier produit commercial de Postscript est Apple LaserWriter en 1985. Il est devenu un standard dans le domaine de l'impression Laser. Préféré par les constructeurs (IBM, DEC, Apple...), utilisé par les développeurs (Microsoft, Digital Research, ...), il fait une entrée en force dans le domaine de la publication assitée par ordinateur (PAO) et de la micro-édition avec, notamment, la LaserWriter d'Apple.

En général, Postscript n'est pas à programmer directement. Il est plutôt un langage de contrôle de machine qu'un langage de programmation de haut niveau. Le programme de PS est souvent généré par l'intermédiaire d'autres programmes d'applications tels que TEX, les logiciels de traitement de texte, de dessin. Le code PS est ensuite prêt à être envoyé à une imprimante pilotée par PS.

Les primitives de PS sont comparables à celles des autres langages graphiques comme CORE, GKS, PHIGS, ses utilitaires de fontes standards le rendent encore plus utile. En programmant directement en PS et en profitant de sa grande souplesse, on peut obtenir des effets étonnants qu'on ne pourrait pas avoir avec d'autres langages graphiques. Un **texte** et un **dessin** peuvent être placés dans n'importe quel sens, position, et avec n'importe quelle taille sur la page; on peut faire presque ce qu'on veut. Mais on ne peut pas être paresseux, car il n'est pas aussi simple et pratique que les autres logiciels de traitement de texte ou de dessin, dans lesquels on n'a qu'à cliquer sur la souris.

B.2 PostScript et la cartographie

La partie graphique de ce langage offre de grandes possibilités à la cartographie, bien qu'à l'origine ce langage n'a pas été conçu pour cette discipline.

On avait cherché en vain des logiciels de dessin pour la représentation des voies routières, c'est finalement PostScript qu'on a retenu tant pour illustrer les opérations de la généralisation cartographique que pour rédiger cette thèse. La plupart des figures dans cette thèse, notamment en ce qui concerne la cartographie, sont produites par des programmes en PS, et les cartes réelles sont réalisées par des codes PostScript générés par le module PS dans le système de généralisation: elles sont aisément insérées dans les fichiers LaTeX pour cette thèse.

Les concepts de mise en oeuvre sont d'une grande simplicité. PS est indépendant des caractéristiques de l'imprimante et voit son avenir assuré: des imprimantes PostScript couleurs sont déjà commercialisées, les écrans PostScript sont à l'études et promettent d'être de puissants outils graphiques. Il est même question qu'un système de fenêtrage basé sur PS (XNeWs par exemple) remplace ceux qui sont basés sur bitmap en raison de la grande puissance de PS qui est, notamment indépendant de la résolution de la machine.

Même le coté *opaque* de PS, généralement considéré comme son grand défaut, nous facilite la tâche pour construire notre réseau routier, et en particulier pour tracer les signes conventionnels de différentes routes de catégories diverses.

B.2.1 Modèles d'image

Les modèles comprennent:

- **SPACE**: le système de coordonnées (x, y), l'unité de mesure est le `point` (72 points = 1 inch). Il existe 3 opérations pour changer ce système:
  1. translation de l'origine. `40 40 translate`
  2. rotation des axes: `30 rotate`
  3. changement d'échelle: `2 1 scale`
- **PAGE**: vide au debut, il sera remplie par les différents opérateurs de peinture qui marquent effectivement sur la page. `fill` pour remplir un *path* avec une couleur donnée, `stroke` pour tracer le contour d'un *path*.
- **PATH**: déclaré par l'opérateur `newpath`, il est composé d'un groupe des points reliés ou non, lignes, polygones, ou de forme quelconque. Le PATH pourra être:
  1. `stroke`: peindre sur la page courante le contour du *path* courant.
  2. `fill`: remplir l'intérieur d'un path fermé avec une couleur.
  3. `clip`: ce path sera le contour de limite (`clipper`) pour le path suivant.
- **CLIPPING PATH**: une frontière. Seuls les éléments de l'intérieur de cette frontière sont effectivement tracés, le reste est ignoré. Par défaut, c'est la limite du papier de l'imprimante. On peut le changer en une forme qu'on veut en en définissant un nouveau. Le clipping path sera l'intersection de toutes les définitions de clipping paths actifs, donc il est de plus en plus petit.

B.2.2 Style de programmation

Un tiers du langage est consacré aux graphiques, le reste constitue un langage de programmation comme les autres langages informatiques. C'est un langage proche du FORTH, assembleur, possédant un dictionnaire et une pile:

- **notation postfix**: les opérateurs se trouvent après les opérandes. L'expression $2 + 3$ sera exprimée comme:

  ```
  2 3 add
  ```

- **pile**: last in first out: le resultat de

  ```
  1 2 3 add   sera   1 5
  ```

- **dictionnaire**: pour stocker des variables et leur valeurs, des procédures et leurs définitions.

- **variable**: la variable $x1$ sera précédée par / et définie par une affectation d'une valeur avec l'opérateur **def**:

  ```
  /x1 2 def
  ```

- **Procedure**: définie avec **def** par un groupe d'opérations entre { } qui ne sont pas tout de suite exécutées. La procédure de changement de **milimètre** en **point** sera:

  /mm {2.835 mul} def.

processus minimal

Un processus minimal d'un programme en PS:

- initialiser un *path*
- placer le point courant.
- déplacer un point.
- tracer un élément.
- terminer.

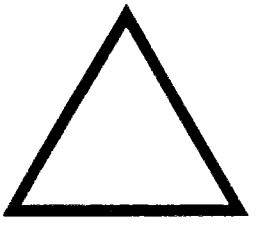
et c'est tout.

B.3 Exemples de constructions des graphiques

Exemple1: tracer un triangle

Tracer un triangle de 30mm (85 points). Après avoir initialisé un path pour ce triangle, on commence par tracer sa base horizontale, ensuite on fait tourner les axes du système de coordonnées de 120 degrés dans le sens direct; on trace un côté; et enfin on ferme ce path.

```
    newpath                 % initialiser
 0   0 moveto        % point initial
85   0 lineto        % 1ieme ligne horizontale
  120  rotate        % tourner les axes (120 degree)
85   0 rlineto       % 2ieme ligne
closepath            % fermer le path
4 setlinewidth       % largeur de ligne
    stroke                  %tracer le path
  showpage                  %imprimer la page
```



Exemple2: tracer plusieurs carrés de couleurs

Cet exemple montre comment définir une procédure en PS et remplir un path avec l'opérateur `fill`.

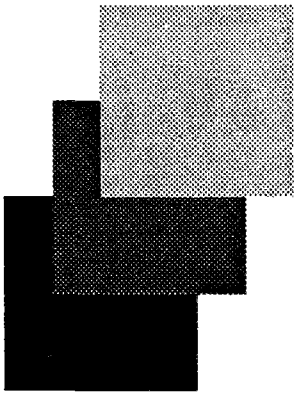
```
%----------define box procedure-------

/box
  {72   0  rlineto
    0  72  rlineto
  -72   0  rlineto
 closepath} def

%-----------Begin programme------------

     252 324 translate

  newpath              %carre noir
     0 0  moveto box
     0 setgray   fill
  newpath              %carre gris
    18 36 moveto box
    .4 setgray   fill
  newpath              %carre clair
    36 72 moveto box
    .8 setgray   fill
showpage
```



Etat graphique courant

L'état graphique courant est composé de:

- **path** courant.

- **point** courant.
- **gray** courant.
- **linewidth** courant.
- **font** courant
- **user système** courant.

Certains opérateurs de PS tels que **fill** et **stroke** effacent le path courant. D'autres opérateurs changent l'état graphique en recevant une nouvelle valeur. On a parfois besoin de conserver un état à un moment ou un path pour s'en servir ou pour y revenir après. Par exemple, après avoir construit une route, on veut la tracer sur une première position, on veut ensuite la translater, ou faire une rotation pour tracer une autre position. On doit tout d'abord conserver le **path** de cette route, ensuite, on peut effectuer des opérations telles que la rotation, la translation, etc. Cela est possible grâce aux opérateurs **gsave** pour stocker l'état graphique et **grestore** pour restaurer cet état graphique conservé.

```
%------
  %construction d'un path
%------
 gsave      % stocker l'etat grahique et le path courant.
 stroke     % tracer  l'etat grahique et effacer le path courant.
 grestore   % restorer l'etat grahique et le path courant.
 fill       % remplir l'interieur du meme path
```

On finit par donner un exemple pour tracer des routes avec des *couleurs* (la figure B.1):

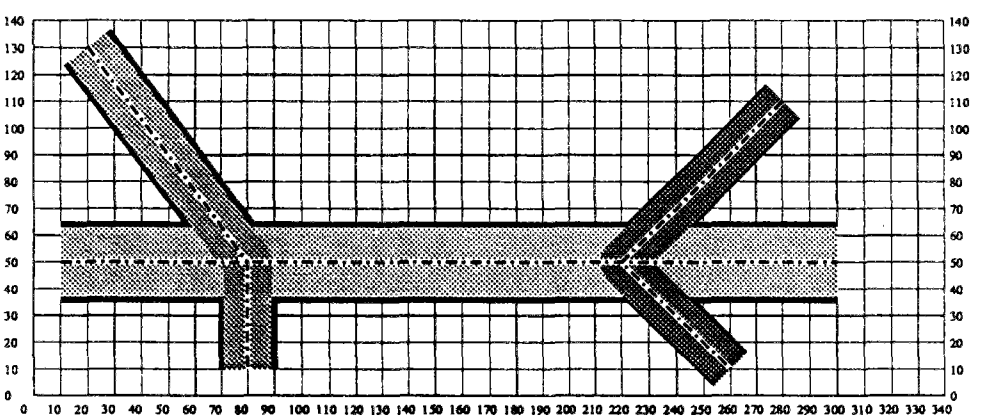


Figure B.1: Tracer des routes avec PostScript

```
%---------------------les def

/saved save def
/MonDict 200 dict def   % define a working dictionary
MonDict begin           % start using it.
   /n-str  10 string def
   /prt-n  {nstr cvs show } def
   /fontxy {/Times-Roman findfont 5  scalefont setfont } def
 /carreau
   { newpath
     /haut exch def
     /long exch def
       fontxy .04 setlinewidth
       0 10 long { /xy exch def
                    xy         0 moveto    0  haut rlineto
                    xy  5 sub -7 moveto xy prt-n} for
       0 10 haut { /xy exch def
                     0       xy moveto  long  0 rlineto
                   -11  xy 2 sub moveto xy prt-n
                   long xy  moveto 2 -2 rmoveto xy prt-n} for
     stroke } def

  /put-xy    {aload pop} def
  /tracer-route    % ----------array des pts: ary-r
       { /ary-r exch def
            ary-r 0 get put-xy moveto
            ary-r aload length 1 sub array astore
            exch pop
            {put-xy lineto} forall
        stroke } def

%---------les (x, y)
  /p1 [10 50] def
  /p2 [80 50] def
  /p3 [20 130] def
  /p4 [80 10 ] def
  /p5 [220 50] def
  /p6 [280 110] def
  /p7 [260 10] def
  /p8 [300 50] def

%-----  programme
 1 setlinejoin
 340 140 carreau %-----tracer des carreaux

   30 setlinewidth [p1 p8] tracer-route
   22 setlinewidth [p3 p2 p4] tracer-route
   18 setlinewidth [p6 p5 p7] tracer-route
```

```
  .8 setgray 26 setlinewidth  [p1 p8] tracer-route
  .6 setgray 18 setlinewidth  [p3 p2 p4] tracer-route
  .4 setgray 16 setlinewidth  [p6 p5 p7] tracer-route

   1 setgray     2 setlinewidth
     [p1 p8] tracer-route
     [p3 p2 p4] tracer-route
     [p6 p5 p7] tracer-route
   0 setgray  [4 2 1 2] 0 setdash .5 setlinewidth
     [p1 p8] tracer-route
     [p3 p2 p4] tracer-route
     [p6 p5 p7] tracer-route
showpage end clear saved restore
```

Annexe C

Fichiers des règles

```
(print
  " * Charger le fichier de Base de Regles: B2newBdr.MI1 : * ")

; regle avec variables
; detecter les relations symetriques entre les Sommets
; completer toutes les relations deja connues

(setq Rg1
       (regle
           'Rg1                      ; le nom de la regle
           '2                        ; le nb-predicat
           '0                        ; le nb-predicats inconnus
           '((symetrique ?R)         ;les premisses
             (?R ?obj1 ?obj2))
           '((ajoute (?R ?obj2 ?obj1))) ; la conclussion
           '()))                         ; non-utilisee


; detecter les besoins de deplacement des Sommet-carf

(setq Rg2.1
       (regle
           'Rg2.1
           '2
           '0
           '((V-c ?s1 ?s2)
             (avec (and (D12<dmd ?s1 ?s2)
                        (S1>S2? ?s1 ?s2))) ; S1 prioritaire
             (non (Ecarter-g ?s2 &- ?-))
             (non (Aconfondre ?s2 ?-))
             (liaison (?delta-dis (dmdvc-dis12 ?s1 ?s2)))
                                       ;la distance a ecarter
             (avec (> ?delta-dis delta-erreur)) ; > precision
            )
           '((ajoute (Ecarter ?s2 ?s1 ?delta-dis))
```

```
          )
          '()))
 ;
;; S1>S2
(setq Rg2.2
      (regle
          'Rg2.2
          '2
          '0
          '((V-c  ?s1 ?s2)
            (avec (and (D12<dmd ?s1 ?s2)
                       (S1>S2? ?s1 ?s2)))
            (liaison (?dis1 (dmdvc-dis12 ?s1 ?s2)))
            (Ecarter-g ?s2 (&x) ?dis2)
            (avec (not (member ?s1 &x)))
            (liaison (&x+ (append1 &x ?s1)))
            (liaison (?delta-dis (max ?dis1 ?dis2)))
           )
          '((ajoute   (Ecarter-g ?s2 &x+ ?delta-dis))
            (supprime (Ecarter-g ?s2 (&x) ?dis2 ))
           )
          '()))



; detecter les besoins de deplacement des Sommet-carf
; S1=S2, on l'enleve

(setq Rg2.3
      (regle
          'Rg2.3
          '2
          '0
          '((V-c ?s1 ?s2)
            (avec (and (D12<dmd ?s1 ?s2)
                       (S1=S2? ?s1 ?s2))) ; S1 prioritaire
            (non (Ecarter-g ?s2 &- ?-))
            (non (Aconfondre ?s1 ?-))
            (non (Aconfondre ?s2 ?-))
            (liaison (?delta-dis (* .5 (dmdvc-dis12 ?s1 ?s2))))
            (avec (> ?delta-dis delta-erreur)) ; > precision
           )
          '((ajoute (Ecarter ?s2 ?s1 ?delta-dis))
            (ajoute (Ecarter ?s1 ?s2 ?delta-dis))
           )
          '()))


;;;;l'ecartement  pas a la limite DMD
;;; entrer le premier Aconfondre
;;; S1>S2
(setq Rg8.1
```

```
       (regle
           'Rg8.1
           '2
           '0
           '((V-c ?s1 ?s2)
             (avec (and (not (D12<dmd ?s1 ?s2))
                        (S1>S2? ?s1 ?s2)))
             (non (Aconfondre ?s2 ?-))
            )
           '((ajoute (Aconfondre ?s2 ?s1))
             (supprime (V-c ?s1 ?s2))
             (supprime (V-c ?s2 ?s1))
            )
           '()))

;;; S1>S2

(setq Rg8.2
       (regle
           'Rg8.2
           '2
           '0
           '((V-c ?s1 ?s2)
             (avec (and (not (D12<dmd ?s1 ?s2))
                        (S1>S2? ?s1 ?s2)))
             (Aconfondre ?s2 ?s3)
             (avec (or  (and (nequal ?s3 ?s1)
                             (not (S1=S2? ?s2 ?s3))
                             (> (dmdvc-dis12 ?s2 ?s1)
                                (dmdvc-dis12 ?s2 ?s3)))
                        (S1>S2? ?s1 ?s3))) ;rapproche au +important
            )
           '((ajoute (Aconfondre ?s2 ?s1))
             (supprime (Aconfondre ?s2 ?s3))
             (supprime (V-c ?s2 ?s1))
             (supprime (V-c ?s1 ?s2))
            )
           '()))

;;; supprime les faits abusifs de V-c
(setq Rg8.3
       (regle
           'Rg8.3
           '2
           '0
           '((V-c ?s1 ?s2)
             (avec (and (not (D12<dmd ?s1 ?s2))
                        (S1>S2? ?s1 ?s2)))
             (Aconfondre ?s2 ?s3)
             (avec (and (nequal ?s3 ?s1)
                        (or (and (not (S1=S2? ?s2 ?s3))
                                 (< (dmdvc-dis12 ?s2 ?s1)
```

```
                                (dmdvc-dis12 ?s2 ?s3)))
                        (S1=S2? ?s3 ?s2)
                        (S1>S2? ?s3 ?s1)))) ;
          )
          '((supprime (V-c ?s2 ?s1))
            (supprime (V-c ?s1 ?s2))
          )
          '()))

;;; S1=S2
(setq Rg8.4
      (regle
          'Rg8.4
          '2
          '0
          '((V-c ?s1 ?s2)
            (avec (and (not (D12<dmd ?s1 ?s2))
                       (S1=S2? ?s1 ?s2)))
            (non (Aconfondre ?s1 ?-))
            (non (Aconfondre ?s2 ?s1))
          )
          '((ajoute (Aconfondre ?s1 ?s2))
            (supprime (V-c ?s2 ?s1))
            (supprime (V-c ?s1 ?s2))
          )
          '()))


(setq Rg8.5
      (regle
          'Rg8.5
          '2
          '0
          '((V-c ?s1 ?s2)
            (avec (and (not (D12<dmd ?s1 ?s2))
                       (S1=S2? ?s1 ?s2)))
            (Aconfondre ?s1 ?s3)
            (non  (Aconfondre ?s2 ?s1))
            (avec (and (nequal ?s3 ?s2)
                       (or (and (S1=S2? ?s3 ?s1) ;soit S=, et +pres
                                (> (dmdvc-dis12 ?s1 ?s2)
                                   (dmdvc-dis12 ?s1 ?s3)))
                           (not (S1=S2? ?s3 ?s1))))) ;soit non S=
          )
          '((ajoute (Aconfondre ?s1 ?s2))
            (supprime (Aconfondre ?s1 ?s3))
            (supprime (V-c ?s2 ?s1))
            (supprime (V-c ?s1 ?s2))
          )
          '()))
```

```
(setq Rg8.6
      (regle
          'Rg8.6
          '2
          '0
          '((V-c ?s1 ?s2)
                          ;l'ecarte pas a la limite DMD
            (avec (and (not (D12<dmd ?s1 ?s2))
                       (S1=S2? ?s1 ?s2)))
            (Aconfondre ?s1 ?s3)
            (avec (or (equal ?s3 ?s2)
                      (and (S1=S2? ?s3 ?s1)
                           (< (dmdvc-dis12 ?s1 ?s2)
                              (dmdvc-dis12 ?s1 ?s3)))))

           )
          '((supprime (V-c ?s2 ?s1))
            (supprime (V-c ?s1 ?s2))
           )
          '()))


(setq Rg9.1
      (regle
          'Rg9.1
          '2
          '0
          '((Aconfondre ?s1 ?s2)
            (Aconfondre ?s2 ?s3)
           )
          '((ajoute   (Aconfondre ?s1 ?s3))
            (supprime (Aconfondre ?s1 ?s2))
           )
          '()))


; 1ere groupement de deplacer un Sommet-Craf

(setq Rg3.1
      (regle
          'Rg3.1
          '2
          '0
          '((Ecarter ?s1 ?s2 ?dis1)  ;les premisses
            (Ecarter ?s1 ?s3 ?dis2)
            (non (Ecarter-g ?s1 &- ?-))
            (avec (nequal ?s2 ?s3))
            (liaison (&s1 (list ?s2 ?s3)))
            (liaison (?dis (max ?dis1 ?dis2)))
           )
          '((ajoute (Ecarter-g ?s1 (&s1) ?dis))  ; les conclusions
            (supprime (Ecarter ?s1 ?s2 ?dis1))
```

```
            (supprime (Ecarter ?s1 ?s3 ?dis2))
            (supprime (Action (Ecarter ?s1 ?- ?-)))
           )
          '())) ;  non-utilisee
;

; grouper les restes

(setq Rg3.2
      (regle
          'Rg3.2
          '2
          '0
          '((Ecarter ?s1 ?s2 ?dis1)          ;les premisses
            (Ecarter-g ?s1 (&x) ?dis2)
            (avec (not (member ?s2  &x)))
            (liaison (&x+ (append1 &x ?s2)))
            (liaison (?dis (max ?dis1 ?dis2)))
           )
          '((ajoute (Ecarter-g ?s1 (&x+) ?dis))
            (supprime (Ecarter ?s1  ?s2 ?dis1))
            (supprime (Ecarter-g ?s1 (&x) ?dis2))
            (supprime (Action (Ecarter ?s1 ?- ?-)))
            (supprime (Action (Ecarter-g ?s1 &- ?-)))
           ) ; la conclussion
          '())) ;                          ; non-utilisee



(setq Rg3.3
      (regle
          'Rg3.3
          '2
          '0
          '((Ecarter ?s1 ?s2 ?dis1)   ;les premisses
            (Ecarter-g ?s1 (&x) ?dis2)
            (avec (member ?s2  &x))
           )
          '((supprime (Ecarter ?s1 ?s2 ?dis1))
            (supprime (Action (Ecarter ?s1 ?- ?-)))
           )
           '()))



(setq Rg4.1
      (regle
          'Rg4.1
          '2
          '0
          '((Ecarter    ?s1 ?- ?-)   ;les premisses
```

```
            (Aconfondre ?s1 ?-)
           )
           '((supprime (Ecarter ?s1 ?- ?-)) ; on confonde d'abord
            (supprime (Action (Ecarter ?s1 ?- ?-)))
           )
           '()))


(setq Rg4.2
      (regle
          'Rg4.2
          '2
          '0
          '((Ecarter-g  ?s1 &- ?-)    ;les premisses
            (Aconfondre ?s1 ?-)
           )
          '((supprime (Ecarter-g ?s1 &- ?-)) ; on confonde d'abord
            (supprime (Action (Ecarter-g ?s1 &- ?-)))
           )
           '()))


(setq Rg4.3
      (regle
          'Rg4.3
          '2
          '0
          '((Ecarter    ?s1 ?s2 ?dis1)  ;les premisses
            (Aconfondre ?s2 ?s3)
            (avec (dmdvc-dis12 ?s1 ?s3)) ; ?s1 ?s3 voisin
           )
          '((supprime (Ecarter ?s1 ?s2 ?dis1))
            (supprime (Action (Ecarter ?s1 ?- ?-)))
            (ajoute  (Ecarter ?s1 ?s3 ?dis1))
           )
           '()))


(setq Rg4.4
      (regle
          'Rg4.4
          '2
          '0
          '((Ecarter    ?s1 ?s2 ?dis1)  ;les premisses
            (Aconfondre ?s2 ?s3)
            (avec (not (dmdvc-dis12 ?s1 ?s3))) ; ?s1 ?s3 voisin
           )
          '((supprime (Ecarter ?s1 ?s2 ?dis1))
            (supprime (Action (Ecarter ?s1 ?- ?-)))
           )
           '()))
```

```
(setq Rg4.5
      (regle
          'Rg4.5
          '2
          '0
          '((Ecarter-g  ?s1 (&x) ?dis1) ;les premisses
            (Aconfondre ?s2 ?s3)
            (avec (member ?s2 &x)) ; ?s2 dans &x
            (liaison (&x-s2 (remove ?s2 &x)))
           )
          '((supprime (Ecarter-g ?s1 (&x) ?dis1))
            (supprime (Action (Ecarter-g ?s1 &- ?-)))
            (ajoute  (Ecarter-g ?s1 &x-s2 ?dis1))
           )
           '()))


; conclut des actions des deplacement en eloignant des groupes

(setq Rg6.1
      (regle
          'Rg6.1
          '2
          '0
          '((Ecarter-g ?s1 (&x1) ?dis)
            (non     (Action (Ecarter-g ?s1 (&x1) ?dis)))
                            ;  est le mieux
           )
          '((supprime (Action (Ecarter-g ?s1 &- ?-)))
                              ; on detrui le precedent
            (supprime (Action (Ecarter ?s1 ?- ?-)))
                              ;detruit les individuelles
            (ajoute   (Action (Ecarter-g ?s1 (&x1) ?dis)))
                              ;  est le mieux
             )
           '()))



; conclut des actions des deplacement
    en eloignant un seul Sommet:
(setq Rg6.4
      (regle
          'Rg6.4
          '2
          '0
          '((Ecarter ?s1 ?s2 ?dis)
            (non (Ecarter-g ?s1 (&-) ?-))
            (non (Action (Ecarter ?s1 ?s2 ?dis)))
```

```
          )
          '((supprime (Action (Ecarter ?s1 ?- ?-)))
            (ajoute   (Action (Ecarter ?s1 ?s2 ?dis)))
          )
          '()))


(setq Rg7.1
      (regle
          'Rg7.1
          '1
          '0
          '((Action (Ecarter-g ?s1 (&x) ?dis))
          )
          '((execution (ecarterment ?s1 &x ?dis))
            (supprime (V-c ?s1 ?-))
          )
          '()))

(setq Rg7.2
      (regle
          'Rg7.2
          '2
          '0

          '((Action (Ecarter ?s1 ?s2 ?dis)))
          '((execution (ecarterment ?s1 (list ?s2) ?dis ))
            (supprime (V-c ?s1 ?s2))
            (supprime (V-c ?s2 ?s1))
          )
          '()))

(setq Rg7.3
      (regle
          'Rg7.3
          '2
          '0
          '((Aconfondre ?s1 ?s2)
          )
          '((execution (Confondre ?s1 ?s2))
            (supprime (V-c ?- ?s2))
            (supprime (V-c ?s2 ?-)) ;?s2 n'existe plus
            (supprime (V-c ?s2 &-)) ;?s2 n'existe plus
          )
          '()))


(setq
      base_de_regles
         '( Rg9.1
          Rg2.1 Rg2.2  Rg3.1 Rg3.2 Rg3.3
          Rg8.1 Rg8.2 Rg8.3 Rg8.6 Rg8.4 Rg8.5
```

```
            Rg4.1 Rg4.2 Rg4.3 Rg4.4 Rg4.5
             Rg6.4  Rg6.1))

;;; Rg7.1 Rg7.2 Rg7.3ne sont pas dans la liste de regles bdr

;;; Rg2.3(S1=S2?)
;;;
```

Bibliographie

[1] R. Cuenin
Cartographie générale
Tome 1: notions générales et principes d'élaborations.
Tome 2: méthodes et techniques de production
Collection scientifique de l'IGN. Eyrolles, Paris, 1972.

[2] P. Lecordix
Traduction de *Généralisation cartographique.*
La Société Suisse de Cartographie, 1977.

[3] F. Bouille
Un modèle de données simultanément partageable, portable et réparti
Thèse de Doctorat d'état, Université Pierre et Marie curie, Paris, 1977.

[4] F. Bouille
A Structured Expert System for Cartography Based on HBDS.
Auto-Carto VI Proceedings Vol.1 1983.

[5] B. G. Nickerson, H. Freemann
Development of a rule-based system for automatic map generalization
Proceedings 2nd International Symposium on Data Handings, pp 537-556, 1986.

[6] W.A. Mackaness et al.
Towards a cartographic expert system .
Proceedings Auto-Carto London Vol. 1 pp. 578-587, 1986.

[7] J-P. Grelot
Cours de Cartographie Assistée par Ordinateur .
IGN, St.-Mandé, 1985.

[8] M. Pernot
Automatisation de la rédaction cartographique: Synthèse et Réflexions
rapport ENSG-IGN, St.-Mandé, 1988.

[9] R. Voyer
Moteurs de Systèmes Experts
Eyrolles, Paris, 1987.

[10] J.M. Hullot
CEYX-Version 15 II: Programmer en Ceyx
Rapports Techniques de l'INRIA, 1985.

[11] J. Chailloux, et al.
LE_LISP de l'INRIA version 15.2
Le Manuel de référence INRIA, Rocquencourt, 1986.

[12] H. Farreny, M.Ghallab
Eléments d'Intelligence Artificielle
Hermes, Paris.

[13] C. Queinnec
LISP Mode d'emploi
Eyrolles, Paris, 1984.

[14] C. Bally, et al.
Les langages orientés objets
CEPADUES-EDITION, Toulouse, 1987.

[15] Masini G., et al.
Les langages à objets
InterEdition, Paris, 1989.

[16] R. Carnap
Introduction to Symbolic Logic and its Applications
Dover Publications, Inc., New York, 1958.

[17] G. Pascal
Mémento de philosophie
Guides Pratiques Bordas, Paris, 1975.

[18] R. Lalement
Logique, réduction, résolution
Masson, Paris, 1990.

[19] D.S. Touretzky
The Mathematics of Inheritance
Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.

[20] W.A. Mackaness, P.F. Fisher, G.G. Wilkinson
Towards a Cartographic Expert System
AutoCarto 6 Vol.1, pp 578-587, London, 1986.

[21] W.A. Mackaness, P.F. Fisher
Automatic recognition and resolution of spatial conflicts in cartographic symbolization
Auto Carto 8 pp 709-718 1988.

[22] M. Monmonier
Towards a Practicable Model of Cartographic Generalization
AutoCarto 6 Vol.2, pp 257-266, London, 1986.

[23] F. Christ
A Program for the Fully Automated Displacement of Point and Line Features in Cartographic Generalization
Nachrichten aus dem Kartern und Vermessungswesen, serie II n^0 35 pp 5-30 1978

[24] Adobe System Inc.
PostScript Language: Tutorial and Cookbook 1987.

[25] Adobe System Inc.
PostScript Language: Reference Manual 1986.

[26] D.M. Mark
Conceptual basis for geographic line generalization
AUTO-CARTO Proceedings Ninth International Symposium on Computer-assisted Cartography, pp 68-77, Baltimore, Maryland, 1989.

[27] K.Stuart Shea et Robert B. McMaster
Cartographic Generalization in a Digital Environment: When and How to Generalize
AUTO-CARTO Proceedings Ninth International Symposium on Computer-assisted Cartography, pages 56-67, Baltimore, Maryland, 1989.

[28] K. Brassel, and R. Weibel
Map generalization
8ième assemblée ICA, Report on Int. Research & Development in Advanced Cartographic techniques, pp 120-144 1987.

[29] E. Charniak, and al.
Artificial intelligence programming
LEA, New Jersey, second edition 1987.

[30] B.C. Moszkowski.
Executing temporal logic programs
Cambridge University Press, 1986.

[31] B. Pasquier
Cartographie numérique des domaines: structuration, modélisation, algorithmique
Thèse d'état ès Sciences de Paris VI, 1987.

[32] K. Mehlhorn
Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness
Springer-Verlag, 1984.

[33] W. Lichtner
Computer-Assisted Processes of Cartographic Generalization in Topographic maps
Geo-processing 1, pp 183-199, 1979.

[34] W. Lichtner
Locational Characteristics and the Sequence of Computer Assisted Processes
Nachrichten aus dem Kartern und Vermessungswesen, serie II n^0 35 pp 65-75, 1978

[35] R.B. MacMaster
Automated Line Generalization
Cartographica Vol. 24 No 2, pp 74-111, 1987.

[36] P. Titeux
Automatisation de processus complexes nécessitant des données structurées, des raisonnements et des jugements qualitatifs
Cognitiva 87 pp 252-258, Paris.

[37] P.F. Fisher, W.A. Mackaness
Are Cartographic Expert Systems Possible ?
Auto-Carto 8, pp 530-534, 1987.

[38] D.M. Mark
Design Criteria for a Cartographic Expert System
Journées: Les systèmes experts et leurs applications, pp 413-425, Avignon, 1988.

[39] J-P. Tsang, D. Brissaud
Méthodologie de structuration de connaissance: Illustration sur la conception d'un système de CFAO: propel
Revue International de CFAO et Infographie Vol 2 No 3, 1987.

[40] J-C. Muler, R.D. Johnson, L.R. Vanzella
A knowledge-Based Approach for Developing Cartographic Expertise
Proceedings, 2nd International Symposium on Spatial Data Handings, pp 557-571, 1986.

[41] G. Robinson, M. Jackson
Expert Systems in Map Design
Auto-Carto 7, pp 430-439, 1986.

[42] B. Buttenfield
Treatment of the Cartographic Line
Cartographica Vol.22 No.2, pp 1-26, 1985.

[43] R. Weibel, B.P. Buttenfield
Map design for geographic information systems
Proceedings GIS/LIS 88, vol.1, pp 350-359, 1988.

[44] K.E. Brassel, R. Weibel
A review and framework of automated map generalization
International Journal of Geographical Information Systems, forthcoming.

[45] X-C. Zhao
La généralisation cartographique par l'intelligence artificielle.
Cahier du CERMA, ENPC, pp. 91-126. juin, 1988 Paris.

[46] X-C. Zhao
Méthodologie de conception d'un système expert pour la généralisation cartographique.
Journée d'IA au MELTM, pp 105-113, novembre, 1989, Arche de la Défense.

[47] X.-C. Zhao
Représentation des connaissances par approche orientée objet.
Accepté au MICAD 91, Paris.

[48] Roland. Schittenhelm
The problem of Displacement in Cartographic Generalization. Attempting a Computer-Assited Solution
Nachrichten aus dem Kartern und Vermessungswesen, serie II n^0 33 pp 65-74 1976.

[49] V.b. Robinson, A.U. Franck
Expert Systems Applied to Problems in Geographic Information Systems: Introduction, Review and Prospects
Auto-Carto 8, pp 510-519, 1987

[50] Paul Churchland, Patricia Churchland
Les machines peuvent-elles penser?
Pour la Science n^0 149, pp 46-53, Mars 1990.

[51] Barr, Feigenbaum, Kaufman
Le manuel de l'intelligence artificielle
Eyrolles, 1986.

[52] Guillaume Dorbes
Intégration d'une méthode d'analyse de l'information (NIAM) dans une méthode globale d'acquisition de la connaissance pour la création d'un système expert
8ièmes journées internationales. Les systèmes experts et leurs applications, pp 317-342, Avignon, 1988.

[53] M. Gondran
Introduction aux système experts
Eyrolles 1984.

[54] M. Cayrol
Le langage LISP.
CEPADUES, 1983.

[55] W.F. Clocksin and C.S. Mellish
Programming in PROLOG
Springer-Verlag, 1981.

[56] J.L. Lauriere
Intelligence Artificielle, résolution de problèmes par l'homme et par la machine
Eyrolles, 1986.

[57] P.H. Winston and B. Horn
Lisp
Addison-Wesley, 1981.

[58] P.H. Winston
Artificial Intelligence
Second edition, Addison-Wesley, 1984.

[59] Adele Goldberg and Daavid Robson
SMALLTALK-80; the language
Addison-Wesley Publishing Company, 1989.

[60] David Gries
The Science of Programming
Collection: Texts and Monographs in Computer Science
Springer-Verlag, New York, 1981.

[61] J.-M. Bergé, L.-o. Donzelle, V. Olive, J. Rouillard
ADA avec le sourire
Presses Polytechniques Romandes, 1989.

[62] Leon Sterling, Ehud Shapiro
The Art of Prolog
The MIT Press, 1986.

[63] J.G.P. Barnes
Programming in ADA
Third edition, Addison-Wesley, 1989.

[64] Reference manual for the ADA programming language
Alsys, ANSI/MIL-STD 1815, Janvier 1983.

[65] Bjarne Stroustrup
The C++ programming language
Addison-Wesley, 1986.

[66] Tony L. Hanson
The C++ answer book
Addison-Wesley, 1990.

[67] Stanley Lippman
C++ Primer
Addison-Wesley, 1989.

[68] Brian W. Kernighan, Dennis M. Richie
The C programming Language
Prentice-Hall, Inc Englewood Cliffs, New Jersey.

[69] Manuel d'utilisation: Le-Lisp V.15.2 sur SUN
CRIL

[70] SUN Common Lisp 3.0 : User's Guide
INTELLICORP.

[71] KEE 3.0, Software Developement System: User's Manual
INTELLICORP.

[72] R. Kent Dybvig
The SCHEME programming Language
Prentice-Hall, Inc Englewood Cliffs, New Jersey.

[73] G.L. Steele, S.E. Falman, R.P. Gabriel, D.A. Moon, D.L. Weinreb
Common Lisp: the language
Digital Press.