Institut National Polytechnique de Grenoble
ENSIMAG

Université Joseph Fourier
U.F.R
Informatique & Mathématiques Appliquées

DEA D'INFORMATIQUE

Présenté par: Petko VALTCHEV

Systèmes Classificatoires et Construction de Taxonomies

Effectué au Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle
(LIFIA, Projet SHERPA)

Date: Mardi 20 Juin 1995

Responsable de projet: M. Jérôme Euzenat

Jury: M. Farid Ouabdesselam
M. Nicolas Halbwachs
M. Augustin Lux
M. Bernard van Cutsem

Résumé

Les taxonomies jouent un rôle important au sein des modèles à objets. C'est pourquoi il y a eu des multiples essais d'intégration des méthodes capables de construire des taxonomies de manière automatique dans le cadre d'un système à objets. Les procédés de construction sont pour la plupart empruntés à l'apprentissage et leur adaptation est propre à un formalisme concret. Les problèmes qui se posent devant une intégration complète entre les objets et la construction de taxonomies sont dus à deux facteurs. Tout d'abord les méthodes existantes de construction sont basées sur des traitements *ad hoc* pour les différents types de données pris en compte. Par conséquent, elles ne peuvent pas être intégrées dans un système de représentation de type extensible. Cette ouverture vers l'extérieur demande une approche générique envers les types que les méthodes existantes n'ont pas. Un deuxième défaut de ces méthodes est qu'elles sont capables de construire des taxonomies mais elle ne tiennent pas suffisamment compte de l'existence de telles taxonomies sur les domaines. Cette incompréhension du rôle fondamental de la taxonomie, à savoir d'exprimer la structure d'un domaine de façon synthétique, est payée par des traitements lourds et coûteux.

Notre approche envers l'inférence de taxonomies s'articule autour d'une idée-phare: construire des taxonomies en s'appuyant sur celles déjà disponibles sur les domaines manipulés. Pour cela nous nous servons d'un modèle très abstrait de la notion de taxonomie, les systèmes classificatoires. Les systèmes classificatoires sont libres de toute considération qui peut favoriser un type particulier de données ou plus généralement une structure de domaine.

Les systèmes classificatoires ne sont pas a priori adaptés à des objectifs comme les nôtres. Le modèle a nécessité certaines extensions afin de pouvoir supporter son utilisation dans la construction de taxonomies. Ainsi nous l'avons complété avec la définition d'une mesure de dissimilarité dite *topologique*. Celle-ci exploite exclusivement la structure taxonomique sur un domaine, ce qui lui permet de s'intégrer totalement dans le modèle. La deuxième extension importante du modèle est l'introduction d'un opérateur de généralisation.

Le produit final de notre travail est la définition d'un algorithme de construction de taxonomies inspiré des SC. L'algorithme exploite pleinement et uniquement les taxonomies dans les sous-structures des objets en s'appuyant sur les deux notions ci-dessus, la dissimilarité et la généralisation. Par le biais des SC, la démarche de l'algorithme s'applique à tout type de données muni d'une structure de taxonomie (objets, types, etc.). Elle s'applique donc au système TROPES sur lequel l'algorithme a été implémenté.

Remerciements

Je remercie...

Jérôme Euzenat pour son aide et sa patience.

Le jury de bien vouloir examiner ce travail.

François Rechenmann de m'avoir accueilli dans son équipe.

Gilles Bisson pour ses conseils précieux tout au long de ce travail

Ma femme, pour son encouragement et son soutien.

Jérôme, Cécile, Jean-Marc, Nina, Olivier, Jutta, Florence, Isabelle et Sueli qui ont toujours su comment créer une bonne ambiance dans les bureaux.

Ma famille, si loin et si proche ...

TABLE DES MATIÈRES

Introduction.....1
Chapitre I. Représentation de connaissances à objets et systèmes classificatoires 5
1. Modèle de base.....5
1.1. Objet, attributs, valeurs 5
1.2. Classes et instances.....6
1.3. Spécialisation et taxonomie des classes 7
2. Le modèle TROPES 8
2.1. Objets en TROPES 8
2.2. Types..... 10
2.3. Concepts versus ADT..... 11
3. Systèmes classificatoires..... 12
3.1. Notions de base..... 13
3.2. Produit et projection..... 14
3.3. Opérateurs d'appauvrissement..... 15
3.4. Systèmes classificatoires et TROPES 17
4. Conclusion..... 18
Chapitre II. Construction de taxonomies..... 19
1. Classification automatique et catégorisation conceptuelle 19
1.1. Classification en analyse de données..... 19
1.2. Catégorisation en apprentissage automatique..... 20
1.3. Conclusions 21
2. Comment comparer les méthodes..... 21
3. Quelques approches existantes 23
3.1. Méthodes de type "plus proche voisin" 23
3.2. Méthodes de type centres mobiles 26
3.3. Méthodes de formation de concepts 29
3.4. Conclusions 31
4. Conclusion..... 32
Chapitre III. Une mesure de dissimilarité pour les systèmes classificatoires..... 34
1. Principes 34
2. Mesures existantes..... 36
2.1. Mesures numériques 36
2.2. Mesures symboliques 37
3. Dissimilarité topologique..... 38
3.1. Dissimilarité globale..... 39
3.2. Fonction d'écart..... 39
4. Trois cas habituels 44
4.1. Domaines énumérés 44
4.2. Domaines totalement ordonnés..... 45
4.3. Domaines partiellement ordonnés..... 49
5. Généralisation et couverture 50
5.1. Généralisation..... 50
5.2. Couverture..... 51
6. Types construits..... 52
7. Insertion dans TROPES..... 52
8. Conclusion..... 53
Chapitre IV. L'algorithme de catégorisation conceptuelle..... 54
1. Algorithme CLUSTER..... 54
1.1. Formalisme..... 55
1.2. Critère LEF..... 55
1.3. Procédure de partition..... 55
1.4. Adaptation initiale pour TROPES..... 56
2. CLUSTER/SC 57

2.1. Module global…… 57
2.2. Module taxonomie…… 58
2.3. Module partition …… 60
3. Conclusions …… 67
Conclusions et perspectives…… 69
Bibliographie…… 71
Annexe A
Le calcul des prédicats annotés…… 75
Annexe B
La dissimilarité sur les types construits. …… 77
1. Ensembles et listes comme des SC ordinaires…… 77
2. Produits de SC…… 85
Annexe C
Preuve des propriétés du Chapitre III et annexe B…… 86
Annexe D
Un exemple d'application de l'algorithme CLUSTER/SC…… 99

Introduction

Le problème

La notion de taxonomie joue un rôle fondamental en représentation de connaissance par objets. Cette dernière discipline introduit une modélisation des domaines concrets à l'aide des objets. Ainsi, chacun des individus d'un domaine est modélisé par un objet. Les objets sont décrits en termes d'attributs qui représentent les caractéristiques des individus. Les classes modélisent des groupes d'individus et par ce fait elles sont liées à des ensembles d'objets, les instances d'une classe. Les classes sont aussi décrites par des attributs qu'elles partagent avec leurs instances. De par leurs descriptions les classes sont liées par une relation de spécialisation. C'est une relation d'ordre partiel qui respecte l'inclusion entre les ensembles d'instances. Par conséquent elle induit une structure de taxonomie sur l'ensemble des classes, on parle de graphe de spécialisation. Le graphe de spécialisation constitue le squelette d'une base de connaissance à objets. Il permet la réalisation du mécanisme d'héritage et il fournit en même temps le cadre général pour l'opération de la classification d'instances et la classification de classes.

D'habitude la construction d'un graphe de spécialisation est intégrée à la modélisation. Mais dans certains cas les connaissances disponibles ne pas suffisantes pour former des classes et les organiser dans une taxonomie. C'est dans ce cas que l'importance d'une procédure d'inférence de taxonomies sur un ensemble d'individus apparaît clairement. Un autre scénario est aussi possible. Certains systèmes à objets récents permettent de maintenir plusieurs taxonomies de classes sur le même ensemble d'objets. Le mode de représentation sous-jacent, dit *multi point de vue*, est en particulier réalisé dans TROPES [Mariño 93] qui constitue le cadre de développement pour notre travail. A la création de chaque nouveau point de vue on cherche à définir une taxonomie sur celui-ci. Par conséquent, la construction de taxonomies y trouve un champs d'application encore plus large.

C'est au problème d'inférence automatique de taxonomies au sein d'un système à objet que notre travail se rapporte. Pour cela nous nous basons sur le modèle concret de TROPES. Mais les résultats sont facilement généralisables dans le cas d'un autre système de ce genre.

Nous pouvons maintenant formuler le problème de façon générique indépendamment de la terminologie en RCO. Ainsi, nous disposons d'une collection d'individus. Ces individus sont décrits par un ensemble d'attributs. Ce que l'on veut est un ensemble de classes décrites en intension. Ces classes devront être structurées dans une structure taxonomique dont l'ordre respecte l'inclusion entre les ensembles d'individus.

Cet énoncé est illustré par un exemple sur le dessin de Fig.1.

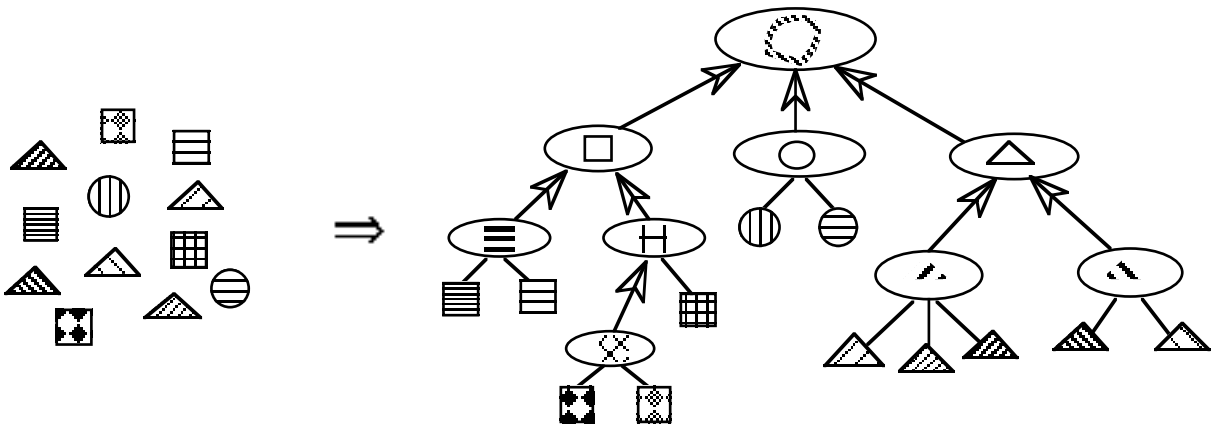


Fig.1. Exemple de taxonomie sur un ensemble d'objets représentant des figures du plan, décrits par deux attributs, `forme` et `fond`.

Les solutions existantes

Les méthodes capables de résoudre notre problème ou des problèmes dérivés ont été développées au sein de deux domaines.

Le première est la classification automatique en analyse de données. Les objectifs de celle-ci varient entre la visualisation, la structuration et la prédiction [Bouch+ 90]. Sa méthode générale consiste à découvrir les régularités dans les données qui sont typiquement introduites sous forme d'une collection d'individus (échantillon d'une population), décrit par un ensemble de traits à valeurs numériques (éventuellement après codage) [Diday+ 82]. La finalité des procédés de classification automatique est un ensemble de classes décrites en extension éventuellement organisées dans une taxonomie pour l'inclusion ensembliste.

La catégorisation conceptuelle, l'autre domaine où trouve des méthodes de construction des taxonomies, s'est développée dans le cadre d'apprentissage comme une version symbolique de la classification. Par rapport à cette dernière, la catégorisation emploie des formalismes beaucoup proches au modèle objet de la RCO. Ses outils sont pour une part hérités de la classification, mais d'autres viennent de la logique, la théorie des graphes, etc. Enfin, la catégorisation partage le souci explicatif de l'apprentissage: les classes construites sont décrites de manière symbolique.

La catégorisation conceptuelle est de loin plus adaptée à la construction de taxonomies au sein de la RCO. Pour cette raison il y a eu des travaux sur l'adaptation de méthodes de catégorisation sur certains formalismes concrets, par exemple le travail rapporté dans [Aguirr 89]. Depuis, les objectifs dans le domaine de la RCO ont évolué. Les systèmes à objets se sont ouvert vers l'extérieur. Ainsi, les utilisateurs peuvent y importer des types de données externes et ceux-ci sont manipulés au même titre que les types propres au système [Cappo+ 95]. Cette tendance d'extensibilité, va à l'encontre des traitements *ad hoc* des types de données qui sont à la base de toute méthode de catégorisation. Ceci constitue un premier problème concret auquel nous avons voulu apporter une solution. Pour compléter, nous pouvons rajouter l'introduction des nouvelles formes de représentation, comme par exemple le mode multi points de vue, qui ne sont pas, bien entendu, prises en compte par les méthodes existantes.

Le deuxième problème se situe dans le champs de la catégorisation. Il est dû au traitement inefficace de la structure dans les objets induite par des relations de décomposition entre les individus modélisés. Les méthodes qui tiennent compte de cette structure en catégorisation appliquent généralement des stratégies d'aplatissement des objets structurés [Stepp+ 86] où proches. Ces stratégies consistent à traiter un objet avec la totalité des informations qui lui sont, de par la modélisation, liées. Le coût de ce genre de traitements est très élevé. Notre conviction est qu'ils peuvent être évités en s'appuyant sur la structure taxonomique.

Notre approche

Notre approche envers les problèmes dus à la richesse et à l'extensibilité des formalismes à objets cités ci-dessus passe par l'introduction des systèmes classificatoires (SC). Ceux-ci constituent un modèle très général de la notion de taxonomie de classes sur un ensemble d'individus. Le modèle permet, en particulier, de s'affranchir des éléments spécifiques à un formalisme de représentation particulier, tels la syntaxe, la structure des classes, etc. Il ne préserve que l'essentiel, à savoir la relation de spécialisation entre les classes (catégories dans le modèle), fondamentale pour la taxonomie, et l'appartenance d'un individu à une catégorie. Par conséquent, les SC constituent un cadre unificateur pour les deux formes de représentation, d'habitude distinguées au sein d'un système: les objets et les types. De plus, les opérations de produit et projection définies sur les SC permettent de modéliser la relation entre les objets, membres des concepts et les valeurs, membres des types abstraits (ADT). Ainsi, un objet est appréhendé comme le produit des valeurs de ses attributs. De même le SC correspondant à un concept peut être considéré comme le produit des SC de tous ses attributs, qu'ils soient des ADT ou d'autres concepts.

L'utilité des SC pour notre problème d'inférence de taxonomies réside dans leur universalité: tout domaine d'individus peut être modélisé en termes de SC. Pour cela un minimum d'information est nécessaire. Ceci permet, en particulier d'étendre les SC sur l'ensemble des domaines gérés par un système à objets, qu'ils soient propres au système ou externes. De plus, les SC modélisent de manière naturelle les notions qui sont susceptibles de poser des difficultés en catégorisation, à savoir les objets composites et les multiples points de vue. L'homogénéité dans la représentation et l'universalité des SC permettent à tout traitement qui ne s'appuie que sur ce modèle de rester générique. Du coup, les problèmes d'adaptation sur un domaine concret se réduisent à la question comment modéliser le domaine en termes de SC. En, particulier, un algorithme d'inférence de taxonomies qui s'inscrit dans le modèle ne rencontre pas les problèmes cités ci-dessus.

Les résultats

Le modèle des SC nécessitait une extension pour pouvoir supporter son utilisation en construction de taxonomies. Après une analyse sur les techniques existants de catégorisation et leur apport potentiels à nos objectifs nous avons mis en évidence les problèmes suivants.

Premièrement, le modèle devait être muni d'un mécanisme de comparaison entre les individus. Ceci est un élément indispensable dans la construction des classes en catégorisation et classification dont le principe fondamental est le regroupement des individus qui se ressemblent. Dans la grande majorité des méthodes effectives cette comparaison est réalisée en termes d'une fonction de (dis)similarité qui donne une expression numérique à la proximité des individus. Il a fallu donc compléter le modèle des SC avec la notion de dissimilarité. A cette fin nous avons défini la *dissimilarité topologique*, une mesure originale, basée exclusivement sur la notion de taxonomie. Cette définition a été motivée par la conviction qu'un lien étroit entre la taxonomie et la dissimilarité existe.

Ensuite, la construction de classes caractérisées de manière symbolique met en jeu des opérations de synthèse des descriptions. Celles-ci relèvent de l'abstraction: on généralise les descriptions des individus afin d'en extraire les caractéristiques partagées. Sur ce point les SC offrent l'avantage de ne considérer aucune description explicite pour une catégorie: elle n'est qu'un produit d'autres catégories. Mais une opération de l'ordre de l'abstraction est tout de même indispensable afin de former les catégories éléments du produit. Ce pourquoi, nous proposons un opérateur dit de généralisation. Comme dans le premier cas, cette nouvelle extension tâche à mettre à profit les taxonomies sur les facteurs d'un produit de SC.

L'algorithme concret

Avec la complétions du modèle des SC la voie était ouverte pour la conception d'un algorithme concret d'inférence de taxonomies. Sa stratégie consiste à construire d'abord les taxonomies sur les concepts primitifs puis sur les concepts qui les référencent et ainsi de suite. Dans cette démarche ascendante, la notion de produit de SC est mise en profit. Chaque concept est vu comme le produit des SC de ses attributs. Ainsi, la construction de la taxonomie sur celui-ci met à contribution les taxonomies sur les facteurs du produit.

Le procédé concret consiste, pour un concept donné, à partitionner l'ensemble d'individus en classes de plus en plus fines. La partition d'une classe en sous-classes est effectuée sur le modèle de l'algorithme CLUSTER [Mich+ 84]. Dans le traitement de chaque facteur du produit (attributs en langage) l'algorithme s'appuie sur les notions citées ci-dessus, en particulier sur la dissimilarité topologique.

Une première version de cet algorithme a été implémentée dans le cadre du système TROPES.

Structure du rapport

Nos travaux ont été exposés sur quatre grandes parties. La première partie (Chapitre I) est consacrée à leur contexte global, la représentation de connaissances par objets et au modèle d'objet concret de TROPES. Nous y introduisons également notre outil générique pour la modélisation des divers domaines, les systèmes classificatoires. Ceux-ci n'étant pas adaptés a priori à la construction de taxonomies, nous présentons ensuite (Chapitre II) une étude comparative des méthodes existantes de construction dont l'objectif est d'en dégager les nécessités fondamentales. A la fin de cette partie nous dressons une liste de primitives à intégrer au sein des systèmes classificatoires. L'intégration effective de ces primitives, dont la plus importante est la mesure de dissimilarité, est décrite dans la partie qui suit (Chapitre III). La définition en termes de systèmes classificatoires est accompagnée des études sur les propriétés que les primitives présentent dans certains cas habituels. Leur utilisation dans le cadre d'un algorithme de construction de taxonomies fait l'objet de la partie suivante. La présentation commence par l'explication de la démarche de CLUSTER dont notre algorithme s'inspire pour s'étendre sur l'architecture complète de ce dernier.

Par souci de ne pas couper le fil du discours nous avons retiré certaines parties intégrales de celui-ci en annexe. Ainsi, l'étude des propriété de la dissimilarité topologique dans les cas de types construits est rapportée dans l'Annexe B. Guidés par le même souci de lisibilité, nous avons regroupé toutes les preuves des propriétés énoncées dans le Chapitre III et l'Annexe B à l'intérieur de l'Annexe C.

Chapitre I. Représentation de connaissances à objets et systèmes classificatoires

Ce chapitre décrit les divers plans du cadre dans lequel notre travail s'inscrit. Dans une première partie, nous cherchons à mettre en évidence les principes fondamentaux de la représentation par objets. Les notions d'objets, de classe et de taxonomie, la relation de spécialisation, etc. sont présentées succinctement. Ensuite, nous adoptons un discours plus concret en nous arrêtant sur les systèmes réels en relation avec notre travail, tels TROPES [Mariño 93] et METEO [Capponi 95]. Cette deuxième partie devrait mettre l'accent sur la richesse des moyens d'expression d'une part et sur l'importance d'un modèle générique pouvant intégrer ces moyens d'autre part. Un tel outil de modélisation, les systèmes classificatoires, est présenté dans la dernière partie du chapitre. Dans la présentation nous introduisons les éléments fondamentaux du modèle tels que la notion d'interprétation, la taxonomie, les opérations de produit et projection. Nous montrons par la suite de quel façon les concepts et les ADT dans TROPES peuvent être modélisés.

A la fin du chapitre, nous formulons de manière plus précise le problème global auquel notre travail devrait apporter une solution.

1. Modèle de base

La représentation des connaissances par objets (RCO) est un paradigme de l'IA qui fait partie de la discipline plus générale de la représentation de connaissances (angl. *knowledge representation*). L'objectif de cette dernière est le développement de formalismes de modélisation de la connaissance. Parmi les critères que ces formalismes devraient satisfaire on pourra citer: la capacité d'être traité par une machine, évidemment, l'intelligibilité pour l'utilisateur, ainsi qu'un bon compromis entre pouvoir expressif et coût algorithmique des traitements associés.

La RCO modélise un domaine à l'aide de la notion d'objet. Les objets sont interprétés comme les individus du domaine et ils possèdent un ensemble d'attributs. Ces derniers modélisent les caractéristiques des individus. Les objets sont regroupés dans des classes qui sont décrites, elles aussi, en termes d'attributs. L'ensemble des classes est ordonné pour la spécialisation, une relation d'ordre partiel qui respecte l'inclusion entre les ensembles d'objets.

Il importe de distinguer le paradigme de la RCO du celui de la programmation par objet (PO) avec lequel il partage un certain nombre de notions fondamentales. Un principe qui met la frontière entre programmation et représentation veut que les objets en RCO aient un aspect déclaratif: ils ne sont pas, dans le cas général, aptes à réagir. Ceci en contraste avec la nature active des objets en PO qui constituent une application en eux-mêmes[Napoli 92].

1.1. Objet, attributs, valeurs

La modélisation en RCO s'articule autour de la notion d'objet. Un objet *décrit une entité du monde, identifiable de façon unique et séparable de son environnement* [Mariño 91]. Les objets n'ont pas une structure atomique. Les caractéristiques des entités qu'ils modélisent y sont intégrées sous forme d'attributs. Un attribut peut être interprété comme une fonction qui projette les objets vers un domaine propre à cet attribut. Ce domaine peut avoir une nature très variée: cardinalité finie ou infinie, présence éventuelle de relations entre ses éléments en

particulier de relations d'ordre, il peut représenter également un ensemble d'objets. Par exemple, si le domaine à modéliser fait partie du marché immobilier et que ses individus présumés sont des habitations, nous pouvons imaginer que parmi les attributs pertinents on pourra trouver l'adresse, le prix, le nombre des pièces, etc.

Les informations propres à un objet que l'on manipule dans le modèle se réduisent à celles exprimées à travers ses attributs, et plus précisément par les valeurs de ces attributs. Ainsi un objet peut être assimilé à la collection des valeurs de ces attributs.

Globalement, on pourra affirmer que l'on est en présence de trois niveaux de description de la connaissance sur le domaine modélisé. On manipule une collection d'*objets*, qui sont décrits par un ensemble d'*attributs* et ceux-ci prennent leurs *valeurs* dans des domaines propres.

1.2. Classes et instances

Lors de la modélisation d'un domaine précis on est toujours amené à considérer deux types de connaissances. Les premières ne concernent qu'un seul individu du domaine à la fois. Celles-ci sont les connaissances *individuelles*. Le reste des connaissances porte sur des groupes d'individus. Elle expriment généralement des propriétés partagées par le groupe voire universelles pour le domaine. On a l'habitude de les appeler les connaissances *génériques*.

La différence entre les connaissances *individuelles* et *génériques* devrait apparaître clairement sur l'exemple suivant. Imaginons que nous sommes en train de constituer une base de connaissance sur le domaine immobilier et que son application sera limité à l'agglomération grenobloise. Maintenant les faits que l'appartement du 9, rue H.Wallon, Fontaine soit un F3 et que la maison à 43, rue de Leningrad soit gérée par l'agence Immo-Fix sont des connaissance individuelles. Au contraire, si nous voulons prendre en compte le fait que dans tout contrat immobilier il y a une clause "prime d'agence", alors nous sommes en présence d'une connaissance générique.

Au niveau modélisation une distinction entre ces deux types de connaissance est faite. Les premières sont modélisées par un type d'objet propre, les *instances*. Généralement les informations concernant un individu sont rassemblées à l'intérieur d'une *instance*. Les connaissances génériques sont, à leur tour exprimées au travers des *classes*.

La distinction entre classes et instances se traduit par une différence structurelle syntaxique. Ainsi les valeurs dans les attributs d'une instance sont des *valeurs simples*, tandis que les attributs d'une classe prennent pour valeur des *domaines*, de sous-ensembles du domaine global de l'attribut sous-jacent.

La figure 2 ci-dessous montre un exemple de classe et une instance appartenant à cette classe.

```
class basic-appartement =
{     appartement-root,
      localisation : domain {"grenoble", "meylan", "eibens", "fontaine"},
      type : domain {S, F1, F2, F3, F4, F5+, M},
      prix : interval [1, 100 000 000]}

object appartement#01 =
{     localisation = "meylan",
      type = F3,
      prix = 800 000}
```

Fig.2. L'exemple d'une instance et une classe.

Le fait que les connaissances modélisées moyennant une classe portent sur un ensemble précis d'individus établit un lien entre la classe et les instances correspondantes. Ce lien est une sorte d'appartenance, il est traduit dans le modèle par la relation *d'instanciation*. Elle lie une instance à "sa" classe et, sous cet aspect, une classe est instanciée par chacune de "ses" instances. La relation d'instanciation entre une classe et ses instances admet deux interprétations:

- *structurelle*: la classe défini un ensemble d'attribut présentés par toutes ces instances, sous ce point de vue la classes constitue une sorte de "moule de fabrication" pour ses instances [Napoli 92],
- *inférencielle*: la classe accueillit toutes les connaissances concernant les valeurs de ces attributs: restrictions de domaines, valeurs par défaut et mécanismes d'inférence, contraintes entre les valeurs, etc.

L'ensemble de toutes les instances d'une classe est une notion importante dans le modèle. Ici aussi, plusieurs façons de définir cette notion sont possibles (cf. le § 4.1. sur les interprétations dans les systèmes classificatoires). Si l'on ne considère que les individus effectivement modélisés, l'ensemble des instances correspondantes forme ce que l'on appelle *l'extension* de la classe. Par contre, si notre considération s'étend sur la totalité d'individus imaginables du domaine, le sous-ensemble obtenu sera *l'intension* de la classe (sur la signification de se mot se référer à [Carré+ 94]).

Les deux notions ci-dessus définissent une correspondance entre les classes dans le modèle et les ensembles d'individus du domaine. Ceci ressemble beaucoup à une fonction d'interprétation pour l'établissement d'une sémantique dénotationnelle. Une relation d'ordre, à l'instar de l'inclusion d'ensembles, est définie entre les classes sur le même domaine. Elle fait l'objet du paragraphe suivant.

1.3. Spécialisation et taxonomie des classes

Nous avons déjà vu que les classes peuvent s'interpréter comme des ensembles d'individus du domaine. De la même façon qu'un ensemble se divise en sous-ensembles, une classe peut être progressivement "éclatée" en plusieurs autres classes qui ne sont instanciées que par une partie des instances de la classe. Les classes obtenues sont appelées des *sous-classes* pour la classe en question. La relation définie ainsi entre la classe et ses sous-classes est dite de *spécialisation*. Le nom exprime le fait que chaque sous-classe définit des connaissances de plus en plus spécifique par rapport à sa *surclasse*. En ce qui concerne l'expression syntaxique des classes, la spécialisation se traduit par l'ajout de nouveaux attributs, la restriction des domaines pour ceux qui existent déjà dans la super classe et/ou cumul de contraintes.

La spécialisation va de pair avec l'inclusion entre les ensembles d'instances: les instances d'une sous-classe sont instances de sa surclasse. On s'aperçoit facilement qu'elle possède les propriétés:

- *transitivité*: les sous-classes d'une sous-classe sont aussi des sous-classes pour une classe donnée,
- *réflexivité*: une classe est toujours sa propre sous-classe ,
- *antisymétrie*: pas de circularité dans les classes.

Autrement dit, la spécialisation est une relation d'ordre partiel sur l'ensemble des classes. Elle permet d'organiser ces dernières dans un graphe orienté et sans circuit dit de spécialisation ou d'héritage. La dernière appellation vient du fait que les sous-classes héritent une partie de leur structure de la surclasse. Ce fait est exploité par le mécanisme d'héritage: il permet de ne stocker dans une classe que les connaissances spécifiques qu'elle apporte vis-à-vis sa surclasse. Les autres connaissances, héritées de toutes les super-classes font objet d'une récupération automatique quand besoin est.

Vu les réflexions ci-dessus, nous pouvons affirmer que le graphe de spécialisation constitue une *taxonomie* (la définition formelle de cette notion est donnée deux sections plus loin dans ce chapitre).

Dans le cas général le graphe de spécialisation n'est représenté qu'implicitement dans une base de connaissances. Ce que l'on peut apercevoir c'est l'indication, pour chaque classe de sa surclasse. Mais la structure de ce graphe est exploitée par la quasi totalité des opérations sur une base à objets. En général, elle permet d'adopter un traitement hiérarchique, avec un degré d'abstraction diminuant. Celui-ci est comparable avec le traitement par indexe dans les BD et représente une réduction logarithmique dans le contexte objet.

Un exemple de taxonomie est donné sur la Fig.3.

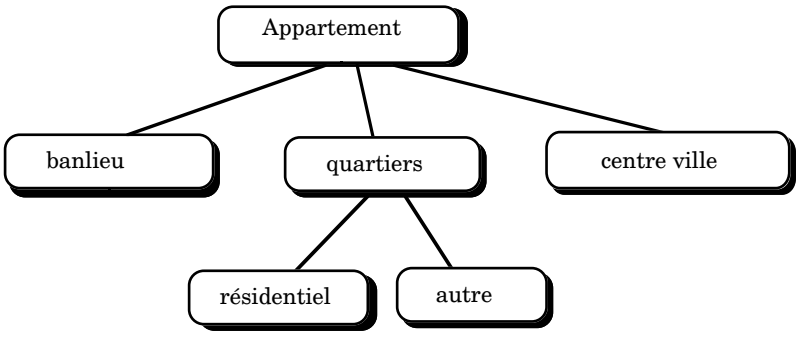


Fig3. Exemple de taxonomie sur le domaine immobilier.

2. Le modèle TROPES

Le système TROPES [Mariño 93, Sherpa 95] est actuellement en cours de développement dans le projet SHERPA. TROPES est un modèle générique qui offre à ses utilisateurs tous les moyens nécessaires pour la création et la gestion d'une base de connaissances à objets. La façon dont cette base est organisée dans TROPES diffère de celle présentée dans la section précédente. Par rapport au modèle de base, des nouvelles notions ont été introduites, dont certaines sont propres à ce système.

Dans TROPES il existe trois moyens pour exprimer la connaissance: les *objets*, les *types* et les *contraintes*. Parmi ceux ce sont les objets qui constituent le moyen le plus intégré. Les deux autres: types et contraintes, permettent d'affiner et préciser les connaissances modélisées moyennant les objets.

Les objets dans TROPES sont partitionnés en grands groupes homogènes, appelés *concepts*. Les objets du même concept se caractérisent par une similarité structurelle, exprimée par le partage des mêmes attributs. Ensuite, les concepts admettent un certain nombre de perspectives sur leurs objets. Celles-ci portent le nom de *points de vue* et définissent un sous-ensemble des attributs des objets visibles. Les points de vue permettent de ne manipuler que les informations pertinentes lors d'un traitement précis des objets. De par la division de l'ensemble des attributs, les classes en TROPES sont définies à l'intérieur d'un point de vue. Par conséquent, les taxonomies sur les objets d'un concept sont aussi propres aux points de vue. Il est donc possible de maintenir des multiples taxonomies sur le même ensemble d'objets. Les notions de concept et de point de vue sont décrites dans le paragraphe 2.1.

Les valeurs primitives comme par exemple les entiers ou les booléens, sont utilisées dans le modèle pour la caractérisation des objets. Elles sont introduites sous forme d'un type abstrait (ADT). Les types abstraits définissent un ensemble de valeurs, le domaine, et des relations et opérations sur ce domaine. Ils constituent une forme de représentation complémentaire aux concepts: les informations manipulées qui ne sont pas exprimées par des objets doivent être typées. La gestion des ADT est confiée à METEO, le système de types intégré à TROPES. La structure et l'utilisation des ADT sont clarifiées dans le paragraphe 2.2.

Finalement, les concepts et les ADT montrent une forte analogie structurelle. En effet, on peut comparer le rôle des objets d'un concept à celui des valeurs primitives dans un ADT. Le même lien existe entre les classes et les types. Le morphisme ainsi défini est discuté dans le paragraphe 2.3.

Voilà comment l'organisation d'une base de connaissances en TROPES peut être résumée. Dans la suite nous allons jeter plus de lumière sur les éléments du modèle ayant un rapport avec l'inférence de taxonomies.

2.1. Objets en TROPES

Conformément au principe énoncé dans la section précédente les individus du domaine modélisé sont représentées dans TROPES par des objets. Les objets sont également décrits en termes d'attributs et les valeurs de ces attributs caractérisent de manière complète un objet. Mais une distinction est faite dans la modélisation entre les groupes d'individus partageant une structure similaire. Ces groupes sont modélisés de façon explicite, moyennant les *concept*.

2.1.1. Concepts

Les concepts peuvent comparer à des grands types d'objets. Ils constituent une partition sur l'ensemble de tous les objets dans une base de connaissances à la TROPES. Cela signifie en particulier que chaque objet appartient à (instancie) un et un seul concept. L'utilité de cette notion se clarifie quand on prend un exemple. Imaginons un domaine où diverses sortes d'entités interviennent dont le traitement dépend de la nature. Pour revenir sur notre exemple d'avant, soient à modéliser maintenant des habitations mais aussi des immeubles entiers, ainsi que des éventuels clients. Une modélisation dans TROPES devrait donc comporter les concepts correspondants: `appartement`, `immeuble` et `client`. Il est clair qu'une distinction dans la modélisation de différents groupes d'entités permettrait, de façon générale, d'augmenter la fonctionnalité de notre base immobilière.

La similitude au niveau structurel à l'intérieur du concept se traduit par l'association entre le concept et les attributs qui décrivent ses objets. En d'autres termes, le concept définit un ensemble d'attributs qui sont universellement partagés par ses instances. Ainsi les attributs sont à interpréter dans TROPES comme des fonctions dont le domaine est le concept sous-jacent et le co-domaine est un ensemble de valeurs associé à l'attribut. Nous discuterons d'avantage sur les types de co-domaines et leurs propriétés dans le paragraphe suivant. Ici notons simplement qu'un tel domaine peut représenter un autre concept de la base de connaissances. Dans le cas général où un objet est caractérisé en termes d'autres objets, c'est-à-dire quand des relations entre les individus du domaine sont représentées, on parle d'*objets complexes*. La modélisation d'un groupe de personnes et des liens familiaux qu'elles entretiennent relève typiquement de l'utilisation d'objets complexes. En TROPES on se propose le traitement d'un cas particulier de relation entre individus, la relation de *décomposition*. Elle a l'avantage d'être une relation antisymétrique et non circulaire. Le type d'objets que l'on arrive à modéliser moyennant cette relation sont appelés *composites* [Mariño 91]. Un exemple d'objet composite dans le contexte immobilier sera la représentation d'un immeuble à condition que ses habitations soient également modélisées.

En tant qu'instances d'un concept les objets présentent, nous l'avons dit, la totalité des attributs définis dans ce concept. Or en fonction du traitement qu'on leur applique, seuls un sous-ensemble de ces attributs peut être pertinent. Prenons l'exemple d'un appartement: la vision sur celui-ci qu'ont ses habitants, l'agent immobilier ou le maçon n'est pas la même. Par ailleurs, la présence d'un certain nombre d'informations hors considération est susceptible d'alourdir inutilement les manipulations. Afin de réduire les informations visibles aux seules caractéristiques pertinentes, on a défini la notion de *point de vue*.

2.1.2. Points de vue

Les points de vue sur un concept permettent d'avoir différentes perspectives sur l'ensemble des instances d'un concept. Chaque point de vue définit un ensemble d'attributs du concept *visibles*. Le reste des attributs est masqué, on ne le manipule pas mais il est présent dans la base. Tout se passe comme si les instances avaient été projetées sur un sous-ensemble de leurs attributs et on ne travaille qu'avec ces projections.

Les classes ont pour rôle d'accueillir les connaissances sur des groupes d'individus. Ces groupes se constituent sur un ensemble de caractéristiques partagées ou proches. Or, les caractéristiques dans TROPES ne se manifestent que sous les points de vues, on comprend pourquoi les classes sont rattachées aux points de vue. Ainsi un point de vue définit, dans le cas général, un ensemble de classes ordonnées, comme nous l'avons vu dans la section précédent, pour la *spécialisation*. Par conséquent les taxonomies sont définies à l'intérieur d'un point de vue.

Il peut y avoir une multitude de points de vue à l'intérieur d'un concept. Ceci détermine la façon dont on représente les objets en TROPES comme *multi point de vue*. Ainsi, sur l'exemple de Fig.4, le concept `appartement` est visualisé avec trois de ses points de vue: `taille`, `prix` et `localisation`. Sur le même exemple, chacun des trois points de vue est muni d'une taxonomie.

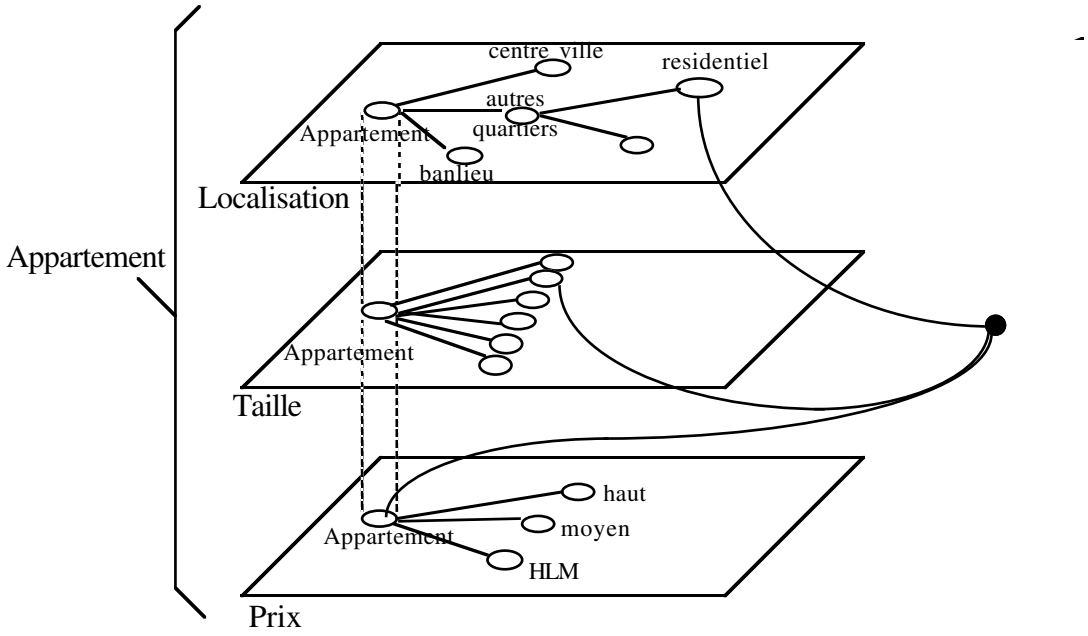


Fig.4. Exemple d'organisation à l'intérieur du concept `Appartement`.

Un objet apparaît sur chaque point de vue de son concept. Et sur chaque point de vue l'objet instancie des classes différentes. Par exemple, un appartement peut appartenir à la classe `quartier résidentiel` sous le point de vue `localisation` et à la classe `très grand` sous le point de vue `taille`. Ceci est le cas de l'instance visualisée par un point sur Fig.4.

Le fait d'avoir de multiples taxonomies sur le même ensemble d'objets constitue un des avantages principaux du modèle. D'un côté, cela permet la représentation de connaissances de provenance variée: les connaissances de plusieurs experts sur un domaine, connaissances acquises de manière automatique et d'autres - de manière manuelle. De l'autre côté le système permet le maintient des liens entre les classes sur différents points de vues. Ces liens, appelés *passerelles* expriment des conditions sur les extensions des classes. Globalement dans un concept, l'ensemble des taxonomies liées exprime une connaissance supérieure à la simple somme sur les points de vue.

2.2. Types

En TROPES tout ne peut pas être exprimé par des objets. Lors d'une modélisation un bon nombre des caractéristiques des individus trouveront une expression au travers des nombres entiers (l'age d'une personne ou le nombre de pièces d'un appartement), réels (le prix) ou encore par des symboles (les noms des villes). Des valeurs de nature plus élaborée telles les dates ou les séquences d'acides aminés, peuvent également être utilisées dans des cas précis. La nécessité de gérer ces valeurs à l'intérieur d'une application en TROPES a provoqué l'introduction des *types abstraits* (angl. *abstract data type*, ADT).

2.2.1. ADT

Les ADT représentent une manière générique d'introduire les types de données, indépendamment de leur implémentation. Ces valeurs deviennent ainsi primitives pour le système qui en fait usage, en l'occurrence TROPES. Elle ne peuvent être manipulées qu'au travers un certains nombre d'opérations fournies au système dans le cadre de l'ADT sous-jacent.

La définition d'un ADT pour un type de données *T* comporte un minimum de relations obligatoire plus un ensemble optionnel d'autres relations et opérations qui varient selon la nature du type *T*.

Les deux éléments indispensables de la définition sont d'abord l'ensemble de toutes les valeurs du type, introduit sous forme d'un prédicat d'appartenance ($\in_T$), puis une manière de distinguer entre deux valeurs non identiques dans *T* matérialisée dans un prédicat d'égalité ($=_T$).

Dans la partie optionnelle, on trouve selon les cas: des prédicats d'ordre ($\leq_T$) permettant de comparer les valeurs, opération de succession ($succ_T$), etc.

2.2.2. Expressions de type et types

Les types de données introduits sous forme d'ADT sont utilisés pour le typage des attributs dans TROPES. Pour cela le langage de description du modèle offre des moyens syntaxiques permettant d'exprimer et restreindre ces types. On y retrouve des *descripteurs de type* tels `domaine`, `intervalle`, `card` ou `except`, ainsi qu'un nombre restreint de *constructeurs*: `un`, `ensemble` et `liste`. Leur rôle est de définir un sous-ensemble de valeurs pour un attribut en jouant sur différents aspects du type. A partir des descripteurs, en effectuant diverses combinaisons, on définit des expressions de type. Ces expressions sont interprétées afin de définir l'ensemble de valeurs qu'elles dénotent. Par exemple pour un type entier l'expression `intervalle` [*0 4*] `except` {*1 3*} a pour interprétation l'ensemble {*0 2 4*}. Il est courant d'avoir deux expressions syntaxiquement différentes ayant la même interprétation. Ainsi l'expression `domaine` {*0 2 4*} a la même interprétation que l'expression ci-dessus.

Afin d'éviter cette ambiguïté, une opération de *normalisation* est effectuée par le système de types METEO. La normalisation traduit les expressions de types dans les classes TROPES sous une forme interne (expressions EOLE, cf. [Capponi 95]).

Après une normalisation, METEO est capable de calculer une *relation de sous-typage* entre différentes expressions de type sur le même ADT. Cette relation est en accord avec l'inclusion entre les ensembles de valeurs correspondants: les domaines. Dans son calcul le système fait appel aux prédicats cités plus haut ($\in_T$ et $=_T$). Les types normalisés sont organisés dans des structures par rapport à la relation de sous-typage. Comme dans le cas de la spécialisation, le sous-typage est un ordre partiel et il respecte l'inclusion. Ainsi les types sur le même ADT utilisant le même constructeur forment un treillis pour le sous-typage. Fig.5 montre un exemple de treillis de types entiers normalisés.

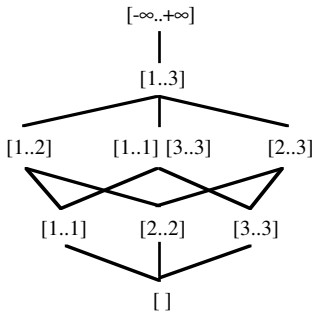


Fig.5. Le treillis des types entiers normalisés qui n'utilisent que les valeurs *1*, *2* et *3*.

2.3. Concepts versus ADT

Les concepts et les ADT en TROPES présentent une structure similaire. Ainsi les valeurs appartenant à un ADT forment un domaine de la même façon que les instances d'un concept définissent un type de donnée particulier. Pour s'apercevoir de ce fait il suffit de remarquer que ces instances peuvent apparaître comme valeurs d'un attribut dans un autre concept. La similitude s'étend au niveau classe. Les classes pour un concept sont la même chose que les types pour un ADT, elles ont une interprétation dans le domaine qui correspond à un sous-ensemble des éléments de celui-ci. L'analogie entre concepts et ADT est exprimée sur Fig.6.

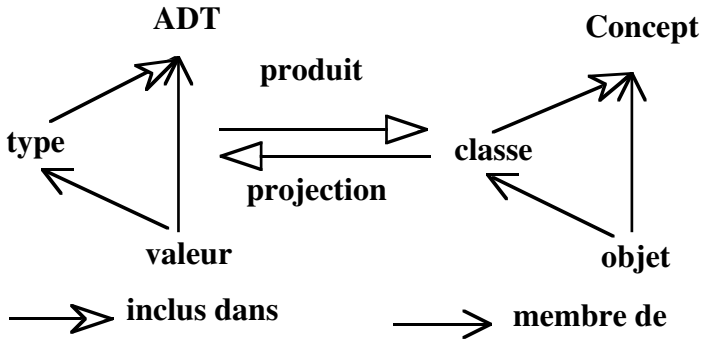


Fig.6. Analogie entre concepts et ADT.

Rappelons que les concepts, au même titre que les ADT, sont des domaines potentiels pour un attribut dans TROPES. Ce fait nous permet de fixer les morphismes qui nous permettent de renvoyer les deux structures l'une vers l'autre. En effet, les concepts peuvent être considérés, à un certain niveau d'abstraction (ici interviennent les contraintes aussi), comme les produits de leurs attributs. Or ceux-ci sont exclusivement des ADT ou d'autres concepts, alors en TROPES les concepts sont des produits de concepts et de ADT. A leur tour les ADT peuvent être considérés comme obtenus par des projections à partir du concept produit. Il est de même avec les concepts qui caractérisent d'autres concepts. Le schéma sur Fig.6 résume la situation. On y trouve l'extension de cette vision "par produit" sur les classes et les objets. Ainsi une classe n'est autre que le produit d'autres classes et de types. Quant aux objets, ils sont considérés comme le produit d'objets et de valeurs. Cette vision particulièrement schématique sur la structure des concepts et des ADT trouve une expression formelle dans le modèle des systèmes classificatoires. Ce modèle est introduit dans la section suivante.

Finalement, mentionnons aux contraintes dont nous avons fait abstraction dans les considérations ci-dessus. Celles-ci constituent le troisième moyen d'expression pour la connaissance dans TROPES. Elles permettent d'exprimer des dépendances entre les valeurs de certains attributs dans les objets. La vérification dynamique de contraintes dans TROPES est à la charge du système MICRO [Gensel 95]. Nous ne parlerons pas davantage du rôle des contraintes dans le système car celles-ci n'ont que très peu d'intérêt pour le sujet qui nous préoccupe.

3. Systèmes classificatoires

L'introduction du formalisme des systèmes classificatoires (SC) répond à un besoin de représentation abstraite pour les manipulations de nature sémantique (classification, catégorisation) sur une base de connaissances [Euzenat 93a]. Pour le cas particulier de TROPES, les systèmes classificatoires apportent l'avantage supplémentaire d'intégrer de façon élégante les notions comme les points de vue, les objets composites, ainsi que les ADT [Cappo+ 95].

Cette section est consacrée à la présentation des SC telle qu'elle a été faite dans [Cappo+ 95]. Au début, nous allons introduire les notions de base, telles la *taxonomie* et le *système classificatoire* (§ 3.1.). Plus loin, les opérations de *produit* d'un ensemble de SC et de *projection* d'un produit seront définies (§ 3.2.). Dans le même paragraphe nous allons également préciser les différents types de produits de SC. Le paragraphe suivant (§ 3.3.) sera la définition des opérations de *dégénérescence* et de *restriction*. Ces deux opérations permettent de

simplifier un SC. Dans le dernier paragraphe nous allons montrer comment les systèmes classificatoires modélisent une base de connaissances dans TROPES.

Les preuves des propriétés énoncées dans la suite de cette section peuvent être retrouvées dans [Euzenat 95].

3.1. Notions de base

La base d'un SC est constituée par deux langages, l'un est le langage des instances L_I, l'autre - celui des catégories, L_C. Les catégories sont à interpréter comme des ensembles d'instances, qui à leur tour sont liées directement aux entités du domaine modélisé. Pour une catégorie c deux sortes d'interprétation sont définies. L'interprétation réelle $I_R(c)$ est l'ensemble des instances que la classe est supposée représenter. L'interprétation abstraite $I_A(c)$ est composée par toutes les entités possibles décrites par la classe. La différence subtile entre les deux interprétations peut être clarifiée par un exemple. Prenons un système classificatoire sur un domaine d'entiers. Le langage L_I.est l'ensemble de tous les entiers de ce domaine et soit le langage des catégories l'ensemble des intervalles. Maintenant considérons les individus qui modélisent les entiers 2,4 et 5. Si l'on cherche la catégorie la plus spécifique dont les trois individus font partie, la seule réponse convenable sera l'intervalle [2,5]. Mais cet intervalle correspond à l'ensemble {2,3,4,5} qui est plus large que notre ensemble recherché {2,4,5}. Nous dirons donc que le premier ensemble constitue l'interprétation abstraite de la catégorie 2,5] tandis que le deuxième - son interprétation réelle. Il est clair que l'interprétation réelle est *a fortiori* inclue dans l'abstraite.

Nous pouvons maintenant définir la notion de taxonomie.

Définition 1

Dans le contexte ainsi défini, une taxonomie, par rapport à une des interprétations, notée I est une structure $(C,\leq)$ où C est un ensemble de catégories, $\leq$ est une relation d'ordre partiel, appelée la *sous-catégorisation*, telle que la propriété suivante est vérifiée:

$$c' \leq c \Rightarrow I(c') \subseteq I(c)$$

La notion de système classificatoire est basée directement sur celle de taxonomie. En gros, un système classificatoire peut être caractérisé comme une double taxonomie.

Définition 2

Un *système classificatoire* $\langle C \leq \ll \rangle$ sur un couple de langages L_C et L_I est la donnée de deux taxonomies: $\langle C \leq \rangle$ relative à $I_R(c)$ et $\langle L_C \ll \rangle$ relative à $I_A(c)$ de façon que:

$$\forall c,c' \in C, c \leq c' \Rightarrow c \ll c'.$$

La relation $\leq$ est dite de *sous-catégorisation*, tandis que $\ll$ porte le nom de *critère de sous-catégorisation.*

Si les éléments d'un SC sont vides on parle d'un *système classificatoire vide*.

Définition 3

Le système classificatoire ‹Ø {} {}› est appelé le système classificatoire vide et il est noté par $\perp$.

3.2. Produit et projection

Les opérations de produit et projection [Euzen 94] permettent d'obtenir un SC à partir d'un autre SC ou ensemble de SC. Pour cela elles modifient essentiellement les langages des catégories et les langages des instances. Ce fait les distingue par rapport aux opérations d'appauvrissement décrites dans le paragraphe suivant.

3.2.1. Produit

Le *produit* d'un ensemble de SC est une opération de composition de structures ordonnées. Elle se base essentiellement sur le produit Cartésien: les langages d'instances et de classes, ainsi que l'ensemble des classes sont définis comme le produit Cartésien des éléments correspondants aux SC de départ. Les deux ordres partiels du produit, $\leq$ et $\ll$, sont aussi définis comme les produits des ordres correspondants[1]. De façon formelle l'opération de projection est définie comme suit

Définition 4

Soit $S_1,\ldots S_n$ un ensemble de systèmes classificatoires (pour $j\in[1,n]$, $S_j=<C_j \leq_j \ll_j>$ est défini sur les langages L_C^j et L_I^j), le produit des systèmes classificatoires $S=\times_{j=1}^n S_j$ est défini par $<\times_{j=1}^n C_j \leq \ll>$ sur les langages $L_C=\times_{j=1}^n L_C^j$ et $L_I=\times_{j=1}^n L_I^j$ de façon que:

$$\forall c=(c_1,\ldots c_n)\in\times_{j=1}^n L_C^j, \forall c'=(c'_1,\ldots c'_n)\in\times_{j=1}^n L_C^j, c \ll c' \Leftrightarrow \forall j\in[1\ n], c_j \ll_j c'_j$$

$$\forall c=(c_1,\ldots c_n)\in\times_{j=1}^n C_j, \forall c'=(c'_1,\ldots c'_n)\in\times_{j=1}^n C_j, c \leq c' \Leftrightarrow \forall j\in[1\ n], c_j \leq_j c'_j$$

Divers types de produits sont distingués *a priori*. Par exemple, dans le cas où les langages des individus $L_I^1,\ldots L_I^n$ sont identiques on parle d'un produit *homogène*. Sinon le produit est dit *hétérogène*. De l'autre côté les produits dont les facteurs sont deux à deux différent sont appelés *disjoints*.

L'ensemble des SC est fermé par l'opération de produit. La preuve de ce résultat peut être trouvée dans [Euzenat 95].

Propriété 1

Le produit d'un ensemble de systèmes classificatoires est un système classificatoire.

L'opération de produit de SC peut être clairement située dans le paysage syntaxique de TROPES. Prenons par exemple un concept `appartement` dont les trois attributs sont `localisation`, `type` et `prix`. D'après les remarques faites dans paragraphe 3.3, le concept `appartement` peut être vu comme le produit des trois ADT correspondant aux trois attributs `localisation`, `type` et `prix`. Au même titre le SC qui modélise le concept est obtenu par produit de SC à partir des SC pour ces ADT.

3.2.2. Projection

[1] Le produit d'ordres est une quantification universelle sur les ordres de départ. En outre, deux éléments de la structure-produit vérifient l'ordre produit ssi toutes les paires d'éléments correspondants à une dimension du produit respectent l'ordre "local".

L'opération de projection est opposée par son effet au produit. Elle permet à partir d'un SC produit d'obtenir un autre produit de dimension inférieure. Autrement dit, cet opération réduit l'ensemble des facteurs d'un produit de SC. En particulier, l'ensemble obtenu peut être vide ou ne contenir qu'un seul SC. La projection est définie dans le modèle de la façon suivante.

Définition 5

Une *projection* π est une fonction qui à partir d'un produit de systèmes classificatoires $\times_{j=1}^{n} S_j$ rend un autre système classificatoire S tel que:

- $S=\perp$, ou
- $S = \times_{j=1}^{m} S'_j (1 \le m \le n); \forall j \in [1\ m], (\exists k \in [1\ n]; S'_j = S_k) \wedge (\forall l \in [1\ m], S'_j = S'_l \Leftrightarrow j = l)$.

.

L'opérateur de projection π est défini sur les deux langages L_I and L_C ainsi que sur l'ensemble de catégories C et les deux relations $\le$ et $\ll$. Certains types de projections sont distingués. Par exemple la projection qui rend un système classificatoire $S=\perp$, est appelée la *projection vide*. Si une projection sélectionne un seul facteur d'un produit on parle d'une *projection unitaire* et on note par σ. Ensuite, on peut comparer deux projections sur le même produit par rapport aux ensembles de facteurs retenus.

Définition 6

Deux projections π_1 et π_2 sont appelées *orthogonales* (noté $\pi_1 \perp \pi_2$) ssi

$$\forall\ S,\ \ \pi_1 S \neq \perp \wedge \pi_2 S \neq \perp \wedge \pi_1 \pi_2 S = \pi_2 \pi_1 S = \perp).$$

Enfin, le résultat d'une projection étant toujours un produit de systèmes classificatoires, celle-ci est une opération de fermeture pour l'ensemble de tous les systèmes classificatoires.

3.3. Opérateurs d'appauvrissement

Les éléments d'un produit de systèmes classificatoires sont, nous l'avons vu précédemment, des produits d'ensembles et d'ordres. Ceci veut dire, en particulier, que l'ensemble des catégories C du produit a une cardinalité énorme (le produit des cardinalités de C_j). Or en réalité un sous-ensemble de celui-ci est considéré. Par exemple, les classes dans un concept n'épuisent pas toutes les combinaisons de valeurs pour les attributs. Juste un petit nombre de telles combinaisons sont "régularisées" sous le statut de classes. De même pour un SC qui modélise (cf. § 4.4.) le concept, des opérations de réduction (ou appauvrissement) sur l'ensemble des catégories C sont à envisager. Ces opérations sont la dégénérescence et la restriction. Contrairement aux opérations de produit et de projection, elle ne modifient ni les deux langages L_C et L_I ni la taxonomie $\langle L_C \ll \rangle$. Leur effet se limite sur la taxonomie $\langle C \le \rangle$.

3.3.1. Restriction

Intuitivement, l'opération de restriction permet de réduire la taxonomie $\langle C \leq \rangle$ sur un sous-ensemble de catégories en préservant toutes les liens de sous-catégorisation entre celles-ci. Une façon moins formelle de présenter cette opération consiste à prendre le graphe G induit par la relation $\leq$ sur C. L'opération de *restriction* est définie par rapport à C', un sous-ensemble de C. Ainsi, la restriction de $\leq$ sur C' (notée $\leq|_{C'}$) correspond au sous-graphe engendré par C'.

Définition 7

Soit $S = \langle C \leq \ll \rangle$ système classificatoire sur les langages L_C et L_I et C' un ensemble de catégories tel que $C' \subseteq C$. La *restriction* de $\leq$ sur C' est la relation $\leq|_{C'}$ déinie comme $\leq|_{C'} = \leq \cap (C' \times C')$.

L'opération de restriction, comme les deux opérations précédentes est une opération de fermeture pour l'ensemble des SC. La propriété suivante est valide (cf. [Euzenat 95])

Propriété 3

La restriction d'une taxonomie est une taxonomie.

Un exemple de restriction sur une taxonomie est donné sur Fig.7. Les taxonomies sont visualisées de façon schématique: la réduction transitive et réflexive de $\leq$ sur l'ensemble de catégories est montré. Le dessin à droite représente la restriction de la taxonomie de départ dessinée à gauche.

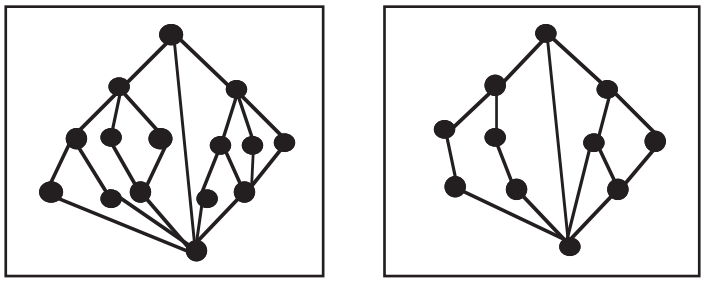


Fig.7. Exemple de restriction sur une taxonomie.

La disparition d'un certain nombre de catégories est visible, ainsi que la préservations des liens existants entre les catégories dans le résultat.

3.3.2. Dégénérescence

L'opération de dégénérescence permet d'obtenir une relation de sous-catégorisation moins rigide sur le même ensemble de catégories C. Par rapport au graphe G défini plus haut, une *dégénérescence* de $\leq$ est un graphe partiel de celui-ci.

Définition 8

Soit $S = \langle C \leq \ll \rangle$ système classificatoire sur les langages L_C et L_I. Une *dégénérescence* $\leq_d$ de $\leq$ est définie comme une sous-ensemble de la dernière relation ($\leq_d \subseteq \leq$) dont la relation binaire est elle-même réflexive et transitive.

L'ensemble des SC est fermé par dégénérescence.

Propriété 4

La dégénérescence d'une taxonomie est une taxonomie.

L'effet de l'opération de dégénérescence sera illustré par un exemple. Sur le dessin de Fig.8 nous avons pris la même taxonomie de départ que précédemment. Sur la droite est dessinée le résultat d'une dégénérescence.

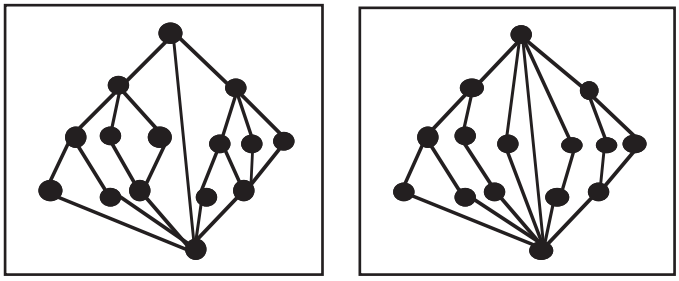


Fig.8. Exemple de dégénérescence sur une taxonomie.

A noter la présence de toutes les catégories d'origine. Les modifications visibles sont l'effet de l'omission d'un certain nombre de couples dans la sous-catégorisation du résultat.

Enfin, pour revenir sur les propos introductifs, notons le lien entre les deux opérations, l'ensemble des catégories et le langage des catégories d'un SC.

Propriété 5

Dans un système classificatoire $<C \leq \ll>$ la taxonomie $<C \leq>$ est une dégénérescence de la restriction de $<L_C \ll>$ sur C.

3.4. Systèmes classificatoires et TROPES

Après avoir introduit les fondements des SC, il importe de préciser leur intérêt pour le contexte de TROPES, mentionné préalablement. Dans la suite nous montrons de quelle façon les ADT et les concepts peuvent être considérés comme des systèmes classificatoires. Le problème étant traité en détails dans [Cappo+ 95] ici nous donnons les résultats principaux.

3.4.1. ADT

Pour un type primitif T, le système classificatoire correspondant est défini sur les deux langages L_I et L_C. Le premier correspond à l'ensemble des valeurs du type, tandis que le deuxième décrit les l'ensemble des types normalisés (cf. § 3.2.2.). L'ensemble des catégories C est le résultat de la normalisation des expressions de type dans les concepts à un moment donné. Les deux ordres partiels sont la restriction de la relation de sous-typage sur les ensembles respectifs: le critère de sous-catégorisation est la restriction sur L_C et la relation de sous-catégorisation - sur C. Les interprétations: l'abstraite, I_A et la réelle, I_R sont équivalentes à l'interprétation des types normalisés dans le domaine de T.

3.4.2. Concepts

La traduction fidèle d'un concept TROPES vers un SC fait apparaître la notion de contrainte. Plus précisément, les contraintes d'un certain profil sont traduites, au même titre que les ADT, vers un SC. Ceci permet de définir le SC pour un concept comme le produit des SC de ses attributs, ADT et autres concepts en général, et ses contraintes. La démonstration du fait que les concepts sont des SC commence par les concepts primitifs, ceux qui ne sont décrits que par des ADT.

4. Conclusion

Dans ce chapitre nous avons précisé les caractéristiques de la RCO, le domaine dans lequel notre travail s'inscrit. Une partie importante de la présentation a été réservée aussi au système TROPES. En particulier, nous avons montré comment celui-ci modélise un domaine donné en se servant des objets et les types.

Nous avons également introduit le modèle des systèmes classificatoires. Ceux-ci s'articulent autour d'une notion de taxonomie très abstraite. Ce fait leur permet de rester génériques dans la modélisation des différents types de données imaginables. Pour notre part, nous avons montré comment les concepts et les ADT en TROPES peuvent être représentés sous forme de SC. La généralité du modèle est le point clé de notre considération: puisque les SC sont universellement applicables, tout traitement qui se limite aux SC comme moyen de représentation hérite de cette universalité. Par conséquent, un algorithme de construction de taxonomies qui travaille sur SC est parfaitement adapté à un système ouvert de types.

Il est temps de préciser comment notre problème de construction de taxonomies dans un cadre à objet se traduit vers les SC. Rappelons que l'objectif du projet est la formulation d'un procédé de construction de taxonomies. Que celui-ci doit être adapté à l'environnement proposé par TROPES et se servir comme formalisme de représentation des systèmes classificatoires. Une formulation possible pourra donc être:

Soit:

- Un ensemble de systèmes classificatoires $S_1,...S_n$. ($S_j=<C_j \leq_j \ll_j>$ et S_j est défini sur les langages respectifs L_I^j et L_C^j où $j \in [1\ n]$).
- Un système classificatoire S défini sur $L_C = \times_{j=1}^{n} L_C^j$ et $L_I = \times_{j=1}^{n} L_I^j$

dont on ne connaît que le critère de sous-catégorisation $\ll$ et $E \subseteq L_I$, un ensemble d'individus.

Rechercher:

- Un ensemble $C \subseteq \times_{j=1}^{n} C_j$ avec la relation de sous-catégorisation $\leq$.

Dans le cas général, on sera amené à traiter d'abord certains éléments du produit afin de pouvoir traiter le produit même. Les taxonomies construites sur ces éléments seront ensuite exploitées dans la construction de la taxonomie du produit. C'est autour de ce principe fondamental que nous allons chercher à définir notre algorithme de construction de taxonomies. Avant d'aborder la description d'un tel algorithme, nous allons examiner dans le chapitre suivantles diverses méthodes déjà existantes. On verra à quel point elles s'adaptent au problème et en quoi elle doivent être modifiées.

Chapitre II. Construction de taxonomies

Après avoir situé notre problème, nous nous tournons vers les domaines où des solutions existent. Notre objectif sera d'examiner ces solutions afin d'en dégager les principes fondamentaux. Une fois la liste de ces notions établie, nous allons conclure sur leur utilité et "faisabilité" dans le cadre des SC.

Dans les sections suivantes nous présentons les grands groupes d'approches existantes en construction de taxonomies. Nous commençons par une brève présentation des disciplines qui offrent aujourd'hui des procédés algorithmiques pour la construction. Ainsi dans la première section nous décrivons les deux paradigmes, le numérique de la classification automatique, et la symbolique de la catégorisation conceptuelle. Dans la suite nous illustrons chacun par quelques exemples de procédés, les procédés numériques et symboliques confondus. Cette présentation n'a pas la prétention d'être exhaustive. A la fin du chapitre nous allons énoncer notre choix de méthode pour l'algorithme à définir.

1. Classification automatique et catégorisation conceptuelle

La procédure générique de construction d'une taxonomie de classes sur un ensemble d'individus porte le nom de *classification* (angl. *clustering*). La classification existe depuis longtemps comme un outil scientifique. Elle permet de structurer un domaine en regroupant les individus dans des classes basées sur des caractéristiques partagées ou proches. Un exemple type de classification est la définition du système périodique des éléments chimiques de Mendéléev. Le gain de la classification est indiscutable: tout d'abord, une intelligibilité accrue de la taxonomie par rapport à la structure plate d'un ensemble surtout si sa cardinalité est importante. La classification a très souvent d'autres objectifs également, par exemple compression des données, visualisation, prédiction. Au fond, il y a une recherche des régularités dans une collection de données. Dans le cas d'un domaine jusqu'à lors mal exploré, la détection de régularités permet de construire des hypothèses. Ces dernières sont appelées à se transformer plus tard en théories générales. Pour reprendre le même exemple du système périodique, il a posé les fondements pour les découvertes sur la radioactivité au XX^{e} siècle[2].

Si l'histoire de la classification est longue, ce n'est que vers les années 60 de notre siècle que la classification est devenue une discipline avec son propre appareil théorique et ses propres outils. Une grande variété de méthodes de classification a vu le jour avec l'arrivée des ordinateurs. A ce jour on compte deux disciplines qui offrent des procédés génériques de classification. Les prochains paragraphes leur sont consacrés. Nous nous efforcerons d'y présenter les motivations, les approches générales, les outils et limites de chacune d'elles.

1.1. Classification en analyse de données

[2] Une appréciation de D.Cohen, trouvée dans [vanCutsem 95]

L'analyse de données (AD) est la discipline au sein de laquelle se sont développées historiquement les premières méthodes de classification. La branche concrète porte le nom de classification automatique.

La classification automatique partage l'appareil théorique de l'analyse de données qui est pour l'essentiel mathématique. Ainsi, les individus, appelés *objets*, sont généralement modélisés de façon numérique. Une collection d'objets est introduite sous forme de tableau objets × variables [Diday+ 82] dont les éléments sont des nombres (réels). Cette modélisation particulière permet de plonger les objets dans un espace numérique de dimension *n*, le nombre des variables, où chacun d'entre eux devient un point. La tâche de la classification automatique devient donc à partitionner le nuage représentant l'ensemble des objets en sous-nuages. Pour ce faire, on se base sur la comparaison entre les objets en termes de fonctions de similarité ou dissimilarité définies sur les "coordonnés" des objets.

Compte tenu de ces caractéristiques, la classification automatique, présente un certain nombre de défauts qui ne lui permettent pas de s'adapter au modèle de la RCO. Dans un premier temps, c'est la restriction sur les types de données pris en compte. Même si d'autres types de données peuvent s'y ramener sous certaines conditions (on utilise dans ces cas des booléens ou des nominaux), la forme numérique se révèle trop rigide dans un grand nombre de cas. En particulier, on ne peut pas exprimer sous cette forme les relations entre les individus [Bouchon+90]. De même pour les domaines à ordre partiel, incompatibles avec les nombres. Enfin, les variables prennent une seule valeur à la fois. Ceci empêche la prise en compte de caractéristiques multi-valuées. Un autre problème se situe au niveau du produit final des méthodes de classification automatique. Les classes produites par ces méthodes sont définies principalement en extension, les descriptions intensionnelles ne sont pas un objectif. Or ces descriptions sont indispensables en RCO.

Malgré les problèmes soulignés ici, les méthodes de classification de l'AD se sont montrées efficaces et robustes dans des cas où la modélisation numérique est parvenue à bien rendre compte de la réalité.

1.2. Catégorisation en apprentissage automatique

La construction de taxonomie peut être retrouvée sous une forme plus proche au modèle de la RCO: la catégorisation conceptuelle en apprentissage automatique. Cette dernière constitue l'équivalent de la classification dans le contexte symbolique de l'apprentissage et de l'IA en général. Par ce biais, les deux disciplines ont en commun une partie de leurs motivations, leurs données et leurs outils.

Sur le point de la représentation, la catégorisation utilise des formalismes de nature symbolique et discrète, pour la plupart issus de la logique (typée). Ceux-ci sont étendus de façon qu'ils puissent intégrer des valeurs de divers types [Larson 77]. Parmi ces formalismes une partie fait référence à la logique du premier ordre (LPO), ce qui leur permet d'exprimer les relations entre les objets. Globalement, des formes d'expression proches aux objets de la RCO sont utilisées [Mich+ 84], [Fisher 87]. Par conséquent, les données en catégorisation constituent un sur-ensemble pour celles décrites précédemment.

Les objectifs de la catégorisation vont au-delà de la simple partition de l'ensemble global des individus en sous-ensembles. On accorde beaucoup d'importance à l'aspect explicatif des résultats. Ainsi les méthodes effectives tâchent de fournir des descriptions symboliques pour les classes construites. Ces descriptions sont très souvent formulées à l'aide du même formalisme ayant servi pour la description des objets.

Les manipulations effectuées sur les données sont de nature discrète. Elles sont basées sur des principes empruntés à un ensemble de domaines comme la logique, la théorie des graphes, la combinatoire, mais aussi la statistique, la théorie de l'information, etc. Les stratégies employées peuvent être globalement caractérisées comme une recherche dans un espace de descriptions.

Il est possible de guider le processus de catégorisation par des informations externes. Ces informations, appelées globalement connaissances du domaine, sont introduites et traitées à part. Les connaissances du domaine peuvent prendre des formes différentes allant d'une indication des types d'attributs [Mich+ 84] à un réseau des objectifs de la catégorisation [Stepp+

86]. Le fait de faire usage des connaissances contextuelles assouplit les méthodes de catégorisation.

Malgré leur plus grande flexibilité sur les données traitées et l'intelligibilité de leurs résultats, les méthodes de catégorisation présentent certains défauts. Par exemple, elles souffrent, comme d'autres branches de l'apprentissage, d'une complexité algorithmique élevée. Celle-ci est due à la nature des procédés employés, pour une grande partie combinatoires. Un autre problème se pose: en effet, les heuristiques utilisées par certaines méthodes pour le parcours de l'espace des solutions ne garantissent pas la convergence ni la stabilité.

1.3. Conclusions

Les problèmes de la catégorisation se sont révélés en quelque sorte complémentaires par rapport à la classification automatique. Ce fait est globalement valide pour l'apprentissage et l'analyse de données. Il y a eu, par conséquent, une forte tendance de rapprochement entre les deux disciplines, afin de réunir le meilleur des deux approches.

L'apprentissage symbolique-numérique est une discipline nouvelle dont l'objectif est de tirer profit d'une coopération entre l'apprentissage et l'analyse de données. Des deux côtés il y a eu des travaux qui ont montré un mariage réussi de techniques symboliques et numériques. Ainsi en catégorisation, qui a démarré déjà comme une discipline symbolique - numérique [Bouch+ 90], on peut citer un système relativement récent, KBG [Bisson 92]. Le système utilise la LPO comme formalisme de représentation. L'élément numérique dans sa démarche est le calcul d'une fonction de similarité entre formules en LPO lors de la construction des classes. La caractérisation des classes par contre est faite par généralisation logique, une opération symbolique. Du côté AD, on peut signaler la multitude des travaux sur les objets symboliques [Diday 93, Gowda+ 94], dont le rôle est d'introduire la notion d'*intension* en AD. Les objets symboliques permettent en particulier de travailler directement avec les descriptions des classes lors d'une analyse dite de *données symboliques*.

Dans la suite nous allons présenter certaines méthodes de catégorisation qui sont par leur nature symboliques-numériques. Nous les avons regroupées avec les méthodes numériques dont ils partagent les stratégies globales de construction (des classes). Ces dernières sont décrites en premier afin de clarifier la démarche. Ensuite nous présentons les parties propres aux méthodes de catégorisation.

2. Comment comparer les méthodes

Avant d'aborder la présentation nous allons introduire quelques notions utiles à la compréhension de ce qui suit.

Concept et classes

Soulignons la différence entre la notion de *classe* (angl. cluster) et celle de *concept* telle qu'elle a été faite dans [Fisher 87]. Contrairement au chapitre précédent, ici par classe nous allons noter un ensemble d'individus. Nous appellerons concept la caractérisation d'une classe exprimée en un langage de description. Tandis que le concept relève d'un formalisme concret, la classe est indépendante de celui-ci.

Ensuite, il nous faut cerner un ensemble d'aspects pour la comparaison des différentes approches en classification. Pour répondre à ce besoin, nous nous tournons vers le travail de Fisher sur le système COBWEB cité plus haut. L'auteur propose de comprendre la catégorisation conceptuelle (angl. *conceptual clustering*) comme une recherche (heuristique) sur trois dimensions:

- l'espace des partitions d'un ensemble,
- l'espace des descriptions d'un ensemble de classes,
- l'espace des hiérarchies de classes sur un ensemble donné.

Partition et caractérisation

Une telle décomposition permet la formulation des principaux problèmes à résoudre par un algorithme de catégorisation. Fisher en cite deux:

- le *partitionnement* (angl. clustering) d'un ensemble d'individus, appelés désormais *objets*, en sous-ensembles "utiles"; l'estimation de l'utilité est confiée à un critère,
- la *caractérisation* en termes d'un *langage de description des concepts* pour chacune des classes trouvées; autrement dit un concept doit être assigné à une classe d'objets.

Une approche de catégorisation concrète se caractérise par les principes des solutions pour ces deux problèmes. Sur ce point, nous allons suivre la méthode de G.Bisson [Bisson 94] qui propose de définir un opérateur générique pour chacun. Ainsi nous allons noter par *F* l'opérateur d'adjonction d'un individu à une classe. La définition d'une caractérisation pour une classe définie en extension sera notée par *G*.

Contrôle

Après avoir explicité les étapes élémentaires de la catégorisation conceptuelle, nous devrons nous intéresser à la façon dont leur application est guidée. En d'autres termes, nous abordons ici un autre aspect sur lequel les approches existantes diffèrent, à savoir le *contrôle* de la recherche sur les trois dimensions.

Il est clair que nous sommes en présence de fortes croissances combinatoires de la taille des espaces des solutions. Ainsi le nombre des partitions possibles sur un ensemble de *n* objets croît si rapidement avec *n* que des recherches exhaustives sont impensables. Dans son article, Fisher parle tout de même de certains cas où les solutions optimales sont trouvées grâce à l'application de techniques de type *branch & bound*. Mais la majorité des approches utilisent des procédés heuristiques de contrôle du style *hill climbing*, *beam search*, ou *tabu search*.

Direction

A l'aspect de contrôle s'ajoute un aspect global de *direction*. Il se caractérise par le principe général de la construction d'une taxonomie. Selon que la transition se fasse du plus spécifique vers le plus général ou dans le sens inverse, on distingue les approches:

- *agrégatives* (angl. bottom-up): la taxonomie est bâtie des feuilles vers la racine, donc en commençant par le plus spécifique et en allant vers le plus général;
- *divisives* (angl. top-down): la tendance est inversée et on va du plus général (la classe racine) vers le plus spécifique (les feuilles).

Certaines méthodes dites de *Formation de concepts* montrent un comportement hybride dans la mesure où elles effectuent localement des transformations *agrégatives* et *divisives*.

Type de la taxonomie

La forme de la taxonomie obtenue à la fin est un autre point de divergence entre les différentes approches en catégorisation.3Son importance pour le processus global de construction a été soulignée dans [Decaest 93]. Les deux formes distinguées sont la hiérarchie stricte et la hiérarchie générale (cf. Fig.9). La première structure est caractérisée par le fait qu'à chaque niveau, les classes constituent une partition sur l'ensemble des objets. Sa définition formelle suit.

Définition 9

Une hiérarchie d'ensembles $\{E_i\}_{i=1,n}$ est définie par la propriété

$$\forall\, i, j \;\; E_i \subseteq E_j \text{ ou } E_j \subseteq E_i \text{ ou } E_i \cap E_j = \emptyset.$$

La deuxième structure ne suit aucune restriction de ce genre. Les classes à un niveau de la hiérarchie peuvent se chevaucher. Elles forment donc, en général, un recouvrement de l'ensemble des objets.

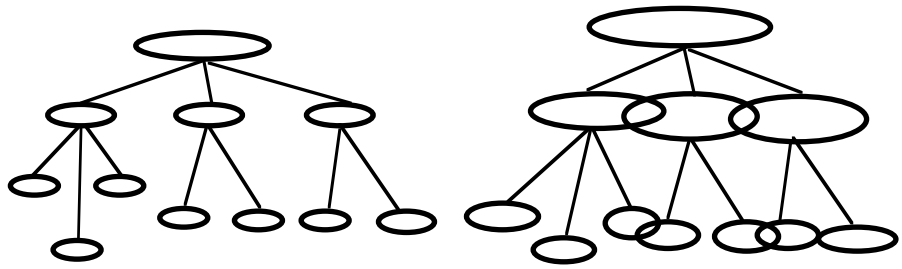


Fig.9. Exemple de deux hiérarchies dont une stricte (à gauche) et l'autre pas (à droite).

Incrémentalité

Notons au passage un aspect pratique lié à l'acquisition des données pour la catégorisation. Pour une partie des méthodes, l'ensemble ***E*** des objets à traiter est supposé connu d'emblée. La taxonomie construite est valide sur la portion de l'univers des individus (angl. *event space*) disponibles (***E***). L'introduction éventuelle d'un nouvel objet provoquera la reprise de la construction à zéro. Ces méthodes sont appelées *non-incrémentales* par opposition aux méthodes *incrémentales* qui sont capables d'acquérir les objets progressivement. A chaque arrivée d'un objet, un algorithme incrémental construit une esquisse de la hiérarchie finale valide sur la portion des objets acquis.

3. Quelques approches existantes

Dans cette sections nous allons présenter synthétiquement des groupes de méthodes existants. Ces groupes ont été formés selon les principes de construction dont nous avons discuté dans la partie précédante. Dans la présentation, les méthodes symboliques sont confondues avec les numériques. Notre conviction est qu'une telle stratégie permet de mieux comprendre la démarche commune ainsi que de cerner les différences dans la réalisation concrète.

3.1. Méthodes de type "plus proche voisin"

Les méthodes de ce type procèdent par des agglomérations de classes successives. Au début, les classes correspondent aux ensembles singletons. Ensuite, à chaque pas un certain nombre de classes, les plus proches par leurs extensions sont agrégées. Un tel procédé suppose l'existence d'une fonction d'évaluation des ressemblances entre les classes. Elle prend la forme d'une similarité si les valeurs élevées correspondent aux proximités fortes ou de dissimilarité dans le cas inverse. Le critère d'agglomération peut porter soit sur le nombre des classes à agréger, soit sur le degré de ressemblance entre les classes.

En ce qui concerne le contrôle, ces méthodes emploient une stratégie globale gloutonne, les choix d'agglomération ne sont pas remis en cause.

3.1.3. Algorithme numérique

C'est un algorithme classique de l'AD. Ses données sont introduites sous forme de tableau individus × variables. L'algorithme se sert d'un nombre fixe des classes agrégées à la fois. Dans la version de base, ce nombre est fixé à deux. La fonction de similarité/dissimilarité varie selon les applications. Elle est essentiellement utilisée pour deux buts. Le premier est le calcul d'un objet virtuel représentant une classe, centre de gravité dans les cas d'une distance euclidienne. Pour les classes singletons ces objets coïncident avec les individus. Ensuite, pour

une classe obtenue par fusion, ils sont calculés comme les centres de gravité. Pour cela des poids, habituellement unitaires, sont assignés aux individus puis hérités par les classes. La deuxième utilisation de la fonction sur les proximités est dans le comparaison de ces objets.

Un exemple d'application de l'algorithme est donné sur Fig.10. Les premières étapes de l'agrégation sont montrées.

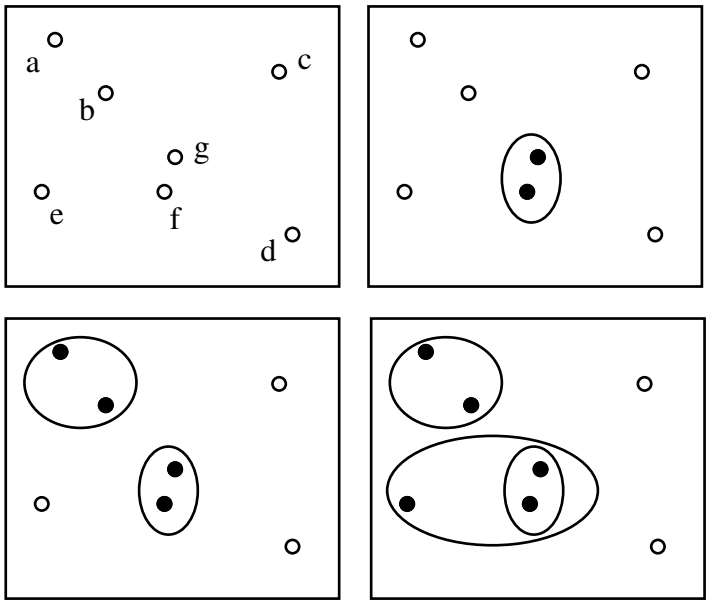


Fig10. Les premières étapes d'une application de l'algorithme de *plus proche voisin*

En conclusion, on peut remarquer que l'opérateur *F* défini précédemment est instancié ici par la fonction d'évaluation des proximités entre les classes. Pour sa part, l'opérateur *G* est le calcul des objets représentant les classes. C'est une forme de caractérisation rudimentaire: les caractéristiques produites n'ont pas une valeur explicative. Leur but est de simplifier le calcul en substituant un point à un nuage de points.

Enfin, nous voulons attirer l'attention sur certaines particularités de la méthode. Tout d'abord, les classes construites sont disjointes. Ce qui explique donc que la hiérarchie obtenue soit stricte. Ensuite, cette hiérarchie est arbitrairement binaire dans le cas de base, car chaque classe est soit une feuille, soit elle possède exactement deux sous-classes. Par exemple, la hiérarchie produite sur les données de Fig.10 est visualisée sur le dessin de Fig.11.

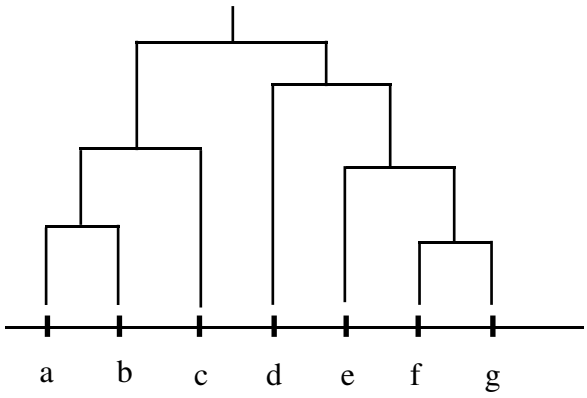


Fig11. Une hiérarchie binaire, construite par l'algorithme du *plus proche voisin*

Une reformulation de l'algorithme ci-dessus a été rapportée dans [Gowda+ 94]. Elle utilise comme langage de représentation les Objets Symboliques, définis par Diday (voir [Diday 93]). Une différences au niveau traitement provient de l'introduction d'objets intensionnels pour représenter directement les classes. Par conséquent la dissimilarité n'est plus définie sur les objets. De plus, chaque agglomération de classes en une nouvelle classe est suivi par la construction de l'objet intensionnel correspondant (par l'application de *Cartesian Join Operator* [Ichino 86]). De cette façon l'opérateur *G* devient explicite dans la nouvelle version de l'algorithme.

3.1.3. KBG

Le système KBG travaille sur des données représentées en LPO (typée). Cette forme de représentation permet le regroupement des individus dans des *observations complexes*. Les observations complexes sont constituées d'un ensemble d'individus et des relations que ceux-ci entretiennent. Un exemple de telles observations est donné sur la Fig. 12.

E1: triangle(A), couleur(A,blanc), carré(B), couleur(B,gris), à-gauche(A,B)

E2: carré(C), couleur(C,foncé), triangle(D), couleur(D,noir), à-gauche(C,D)

Fig.12. L'exemple de deux observations complexes.

L'algorithme de catégorisation de KBG préserve le schéma général d'en haut. C'est-à-dire que les pas élémentaires de la construction sont des agrégations entre classes. Mais cette fois-ci, l'accent est mis sur le comparaison entre observations complexes. Une telle comparaison relève de l'appariement partiel entre les deux graphes (cf. Fig.13). Pour ce faire, une mesure de similarité entre formules en LPO est définie. Elle évalue les ressemblances entre les individus modélisés en fonction de leurs caractéristiques propres, mais aussi de la totalité d'individus en relation avec ceux-ci. La mesure de similarité guide l'appariement et *vice versa*. Par conséquent le meilleur appariement est celui qui optimise la similarité totale entre deux observations.

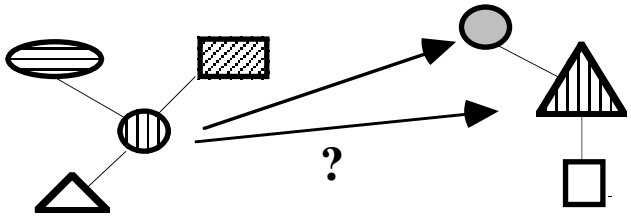


Fig.13. Le problème de l'appariement entre deux observations complexes.

Une fois les similarités calculées, une étape agglomérative, très proche de celle décrite dans le paragraphe précédent, a lieu. Avec une différence essentielle, on agrège toutes les classes dont la similarité est au dessus d'un *seuil* pré-défini. Ainsi, après un pas agrégatif on se retrouve, en général, non plus avec des partitions, mais avec des recouvrements de l'ensemble des observations. Par conséquent, le produit global est une structure de hiérarchie non stricte. La caractérisation des classes obtenues par fusion est une étape indépendante de l'algorithme. Elle suit chaque pas agrégatif. L'opération sous-jacente est une généralisation en LPO. Pour

chaque nouvelle classe elle produit une observation complexe qui subsume les observations des classes constituantes.

L'approche symbolique-numérique de KBG se distingue de celle de l'analyse de données ci-dessus par la formation de classes non disjointes et donc d'une hiérarchie non stricte. Les hiérarchies obtenues ont aussi un facteur de branchement généralement plus important que deux. A noter l'expression naturelle de la structure dans la connaissance par des prédicats en LPO. Comme précédemment, l'opérateur *F* est l'estimation d'une proximité, cette fois-ci une similarité. *G* est une généralisation logique.

3.1.3. Conclusions

De ce premier groupe de méthodes le lecteur devrait retenir les particularités suivantes. Premièrement, les fonctions de similarité/dissimilarité jouent un rôle important dans la construction des classes. C'est autour d'un couple d'objets très proches que la construction de la classe se développe. Ensuite, il convient à remarquer la manière gloutonne et ascendante de construire la taxonomie. Il n'y a pas de retour arrière: une fois les classes constituées, elles ne sont jamais remises en cause.

3.2. Méthodes de type centres mobiles

Les méthodes de ce groupe sont opposées à celles décrites précédemment dans la mesure où leurs stratégies de construction de taxonomies sont descendantes. Elle procèdent par des partitions progressives de l'ensemble de tous les objets. Ceci implique une autre technique pour déterminer les limites d'une classe. En effet, on y trouve dans les deux méthodes présentées ci-dessous, l'importante notion de noyau, un individu autour duquel la classe est construite. Les noyaux des classes ne sont généralement pas connus à l'avance.

3.2.3. Algorithme de base

La définition de l'algorithme des centres mobils est due à Forgy [Forgy 65]. L'algorithme travaille sur des données sous forme de tableau individus × variables. Il applique une technique itérative pour la construction d'une partition. Un pas est associé à un ensemble de noyaux. Les opérations effectuées sont la construction des classes autour des noyaux, puis le choix des noyaux pour le pas suivant à l'intérieur de ces classes. Nous en illustrons le fonctionnement par un exemple (cf. Fig.14).

L'algorithme prend en paramètre le nombre des classes, soit *k*, à construire sur un ensemble d'individus (ici visualisés par des points - Fig.14a). Au pas initial, il tire au hasard *k* objets différents (b). Ceux-ci deviennent les noyaux pour les classes à construire. La construction proprement dite consiste à parcourir la liste des objets restants et d'assigner chacun d'entre eux à une classe. Cette dernière est choisie en fonction du noyau le plus proche (c). Une fois de plus une fonction de dissimilarité, éventuellement une distance, est utilisée pour l'évaluation des proximités entre les objets. Après la fin de la construction des classes (d), on procède au calcul des nouveaux noyaux (e). Ce sont les objets qui, pour une classe donnée, minimalisent la somme de toutes les dissimilarités à l'intérieur de la classe. Après le choix des nouveaux noyaux, une nouvelle distribution recommence. L'algorithme s'arrête quand les noyaux deviennent stables (f). Il est également possible de guider le choix des noyaux par un critère global, afin d'éviter les extrêmes locaux. Par exemple, on peut se servir de l'inertie intra-classe dans les cas où la dissimilarité est une distance euclidienne.

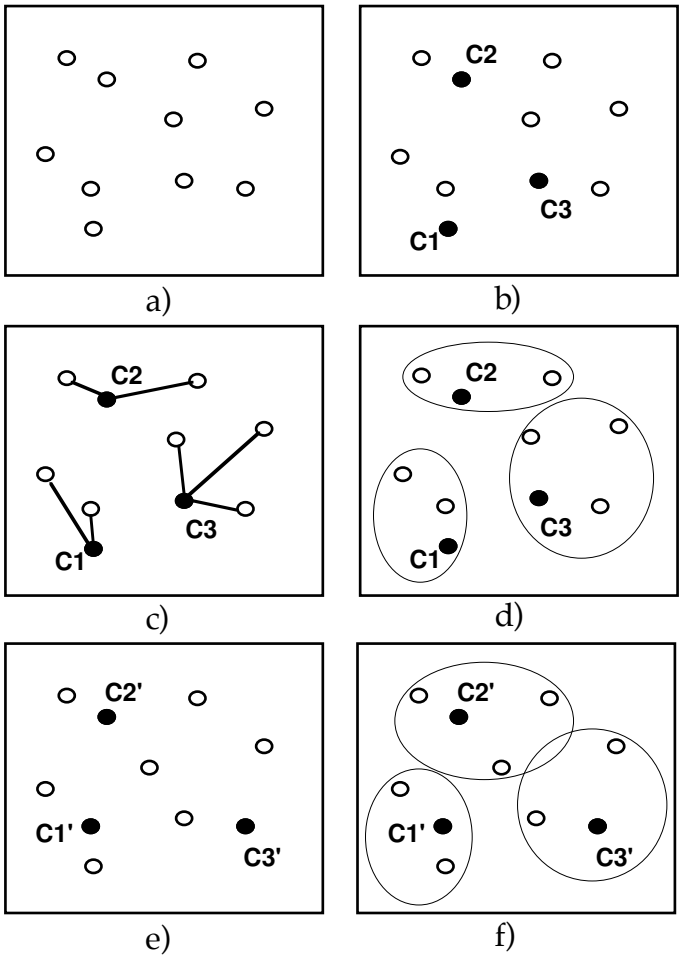


Fig.14. Un exemple d'application de la méthode des centres mobiles.

Une extension de la méthode des centres mobiles dite des *nuées dynamiques* a été proposée dans [Diday+ 76]. La notion de noyau est révisée, elle s'étend ici à un ensemble d'individus. Ces ensembles sont les *nuées* centrales pour les classes à construire. L'introduction de la notion de nuée tente de réduire la dépendance de la partition finale du choix des noyaux au départ. Le contrôle est organisé autour de la notions de pas de base et pas d'essai. Les pas du premier type sont des pas obligatoires à effectuer tant qu'aucune configuration stable des nuées ne soit atteinte. Les seconds sont optionnels, ils ne sont réalisés que s'il y a une amélioration du critère de qualité.

En conclusion de ce paragraphe il faut remarquer que les deux méthodes numériques montrent une utilisation intensive de la dissimilarité. Celle-ci joue le rôle principal dans la construction des classes. L'opérateur *F* concret y repose donc entièrement. Si l'on admet que les noyaux (nuées) constituent une forme de caractérisation, alors l'opérateur sous-jacent serait aussi instancié par le calcul des dissimilarités.

3.2.3. CLUSTER

Le premier algorithme décrit dans ce paragraphe, CLUSTER [Mich+ 84] peut être vu comme l'adaptation des *nuées dynamiques* en LPO et plus précisément en un langage dérivée, APC.

L'algorithme conserve le schéma de fonctionnement de son prédécesseur numérique. On y retrouve le choix aléatoire de noyaux ainsi que la migration de ceux-ci au cours de la construction. La distinction est faite au niveau méthode de formation de classes. Elle présente une caractéristique essentielle, à savoir l'intégration complète de celle-ci avec la procédure de caractérisation des classes.

La formation des classes est organisée en deux étapes. D'abord il y a la construction des nuées (angl. *core*). Pour ce faire, l'algorithme met en jeu une relation de couverture. La couverture est calculée à partir de descriptions très rudimentaires pour les classes. Pour obtenir de telles descriptions on fait appel à une méthode de discrimination, l'algorithme STAR [Mich 83]. A la fin de cette étape, les nuées sont formées et il reste un ensemble d'individus dont l'appartenance à une des futures classes est à établir.

La deuxième étape consiste essentiellement à distribuer ces individus sur l'ensemble des nuées. Pour ce faire les individus sont considérés un par un. L'adjonction à chacune des classes est évaluée par rapport à la qualité de la partition obtenue. Cette situation est schématisée sur Fig.15. Le dessin montre l'exemple d'un objet en cours de traitement. Les lignes pointillées montrent les limites virtuelles des classes induites par leurs descriptions.

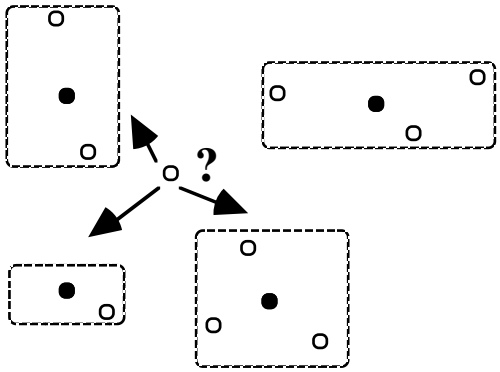


Fig.15. L'état initial d'une étape de la distribution.

A priori toutes les possibilités sont évaluées. Le critère utilisé porte de façon prioritaire sur les descriptions des classes. Il permet de choisir la classe à laquelle l'objet sera assigné par la suite. Suite à l'assignation, la description de la classe sera "étendue" afin de couvrir le nouveau membre. Ceci est illustré sur Fig.16. dont le dessin montre la suite de la situation ci-dessus.

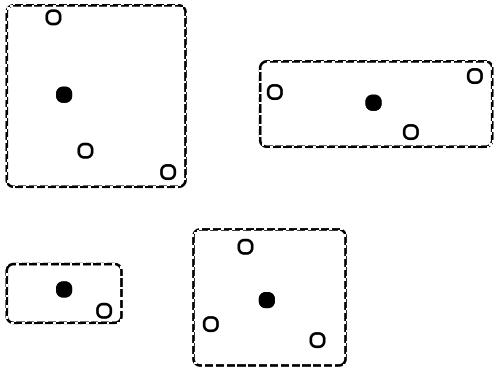


Fig.16. L'intégration de l'individu au sein d'une classe.

L'objet a été intégré à la classe en haut à gauche. La description de celle-ci est modifiée de manière appropriée.

Le fonctionnement de CLUSTER sera discuté plus en détails plus loin dans ce rapport. Ici nous n'avons donné que le minimum nécessaire pour la compréhension de l'approche.

Il est à noter que la formation d'une description, par généralisation, accompagne fidèlement toute opération sur l'extension d'une classe. Le critère d'optimisation qui guide la construction est orienté vers les descriptions plutôt que vers les classes mêmes. Ainsi, un objet n'est ajouté à la classe que si la description obtenue à la suite n'est pas la meilleure sur l'ensemble des classes. Par conséquent, le rôle que joue la dissimilarité n'est qu'un rôle secondaire, elle sert tout simplement à choisir les noyaux pour le pas suivant de l'algorithme.

Ce fait nous permet de conclure que dans le cas de CLUSTER les deux opérateurs, d'adjonction et de caractérisation, sont respectivement instanciés par la couverture et la généralisation.

L'algorithme dans le système ULS [Esposito 94], constitue une version révisée de la méthode de CLUSTER. Cet algorithme utilise le même langage de représentation des connaissances. Une partie des modifications essentielles dans ULS est liée aux notions de noyau et dissimilarité. En première lieu, l'aspect itératif du calcul des noyaux a disparu. Ils sont déterminés par rapport à ce que les auteurs appellent le *prototype du concept*. Celui-ci est une description très générale et partielle, déduite de la connaissance disponible sur le domaine modélisé. La dissimilarité sert ici à déterminer l'individu le plus proche du prototype.

La dissimilarité a changé de fondement et de rôle dans ULS. D'abord, par sa définition, elle mesure un appariement partiel entre un concept et un objet. Ensuite, elle est également utilisée est au sein d'un critère de qualité des classes construites.

3.2.3. Conclusions

Après avoir présenté le deuxième groupe de méthodes, résumons leurs caractéristiques intéressantes. En premier, les classes construites sont disjointes, les concepts le sont, en général, aussi. Ces classes sont construites de manière itérative, autour de centres qui peuvent migrer. Les algorithmes symboliques de ce type tendent à s'affranchir de l'influence de la dissimilarité. Finalement, les taxonomies croissent vers le bas et leur construction est un procédé glouton.

Un important défaut de ce type d'algorithmes est le besoin de connaître *a priori* le nombre de classes de chaque partition. Une alternative en est la comparaison de plusieurs variantes à cardinalité différente pour une telle partition. Ceci peut être un bon remède du point de vue théorique, mais il n'est nullement avantageux pour les performances de l'algorithme.

3.3. Méthodes de formation de concepts

Les systèmes que nous allons présenter dans ce paragraphe ont tous deux traits qui les distinguent de ceux décrits précédemment. D'abord les classes construites ne sont pas manipulées de manière explicite, seuls leurs descriptions, les concepts, sont mémorisées dans ce que l'on appelle un arbre de concepts. Ceci explique l'absence d'algorithme numérique dans la présentation. Ensuite, ces méthodes sont capables d'acquérir les individus à catégoriser au fur et à mesure de la catégorisation. Les algorithmes sous-jacents sont donc incrémentaux. Ce fait joue un rôle décisif sur la stratégie du traitement. Même si la tâche globale peut encore se résumer à la construction d'une taxonomie, un pas élémentaire ne représente plus la construction d'un niveau ou d'une descendance. Ici on procède par l'intégration d'un objet, le dernier arrivé, à l'intérieur de la taxonomie courante, construite sur les objets déjà acquis. L'intégration commence par le concept-racine et continue jusqu'à ce qu'un concept terminal soit atteint. Ce type de procédé est illustré sur la Fig.17. Le dessin doit être déchiffré ainsi: la situation lors de l'arrivée d'un objet est donnée à gauche tandis que le résultat de l'intégration est à droite. Les lignes grasses dans la taxonomie résultante représentent le chemin de la descente effectuée lors de l'intégration. Les exposants symbolisent la modifications des concepts correspondants suite à l'intégration de l'individu.

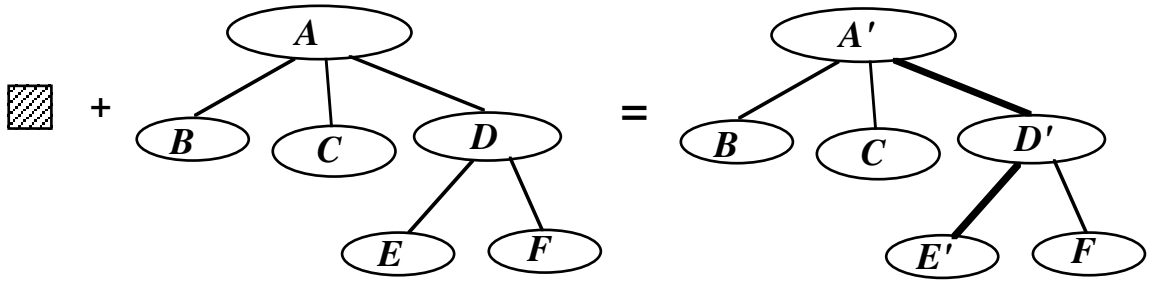


Fig.17. Une étape de la formation de concepts.

Les concepts, dits *probabilistes*, sont constitués de façon particulière. Un concept représente des distributions probabilistes des attributs sur l'ensemble des individus que le concept intègre. Il a la forme d'une liste de triplets (Attribut Valeur Probabilité).

Ce type de catégorisation porte le nom de *formation de concepts*.

3.3.1. COBWEB

L'algorithme COBWEB [Fisher 87] est l'algorithme type de la formation de concepts. Il possède toutes les caractéristiques citées ci-dessus. Sa stratégie est incrémentale, globalement descendante et, à un niveau donné de la taxonomie, consiste à intégrer l'individu dans une partition. L'intégration est le choix d'un concept, la modification de celui-ci de façon à ce qu'il rende compte de l'individu et puis la continuation du processus dans le sous-arbre dominé par le concept. Pour cela on a le choix parmi plusieurs opérateurs d'intégration. Les principaux opérateurs sont:

- *création* d'un nouveau concept singulier et son nœud dans l'arbre.
- *intégration* de l'objet dans un concept existant,
- *éclatement* d'un concept en deux nouveaux dont l'un intègre l'objet,
- *fusion* de deux concepts en un, intègrant l'objet.

L'effet de chacun de ces quatre opérateurs sur un concept de l'arbre est visualisé sur la Fig.18. Les opérateurs sont dessinés dans l'ordre de leur énumération ci-dessus.

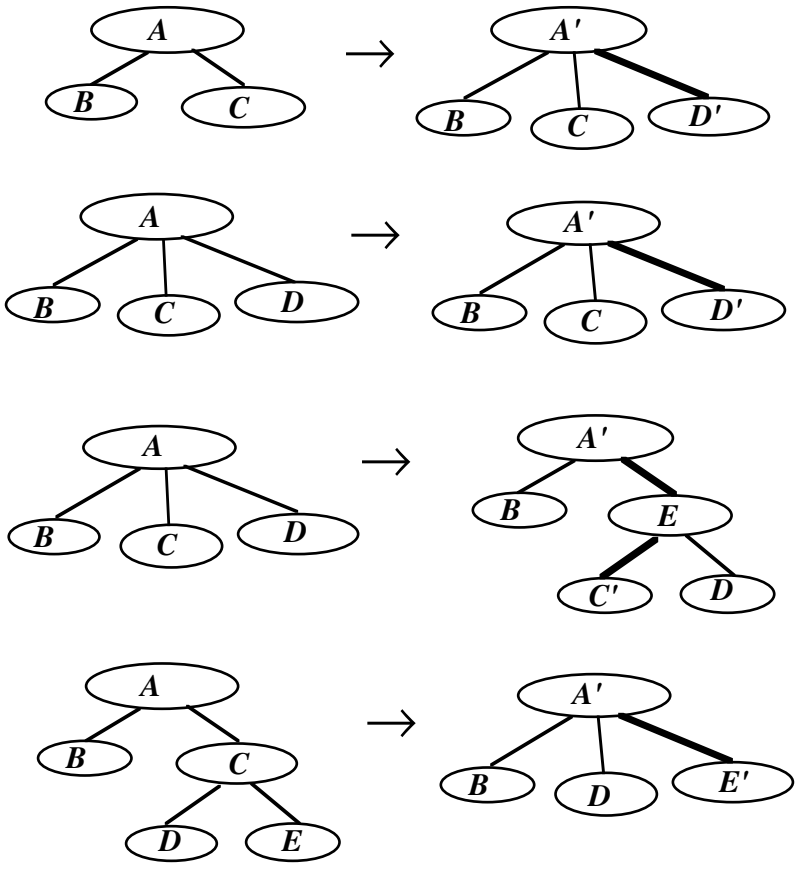


Fig.18. L'effet des opérateurs de *création*, *intégration*, *éclatement* et *fusion*.

L'effet de chacun des opérateurs est évalué selon un critère de qualité afin de choisir le meilleur. Ce critère, appelé d'*utilité de catégorie*, est calculé sur la partition qui résulte de l'application d'un opérateur. Il favorise la faculté prédictive d'une partition. Pour cela des probabilités suivantes sont prises en compte:

- la *prédictivité* d'un couple attribut-valeur $A_i=V_{ij}$ pour une classe C_k, soit $P(C_k \mid A_i=V_{ij})$, c'est-à-dire la probabilité de retrouver la classe d'un objet en connaissant la valeur de l'attribut respectif.
- la *prédictibilité* d'une classe C_k pour un couple attribut-valeur $A_i=V_{ij}$, soit $P(A_i=V_{ij}|C_k)$, c'est-à-dire la probabilité de deviner correctement la valeur de l'attribut pour un objet donné en sachant à quelle classe l'objet est attaché.

Le but est de maximiser les deux probabilités, ce qui a donné l'expression suivante au critère d'utilité de catégorie:

$$\mathbf{CU(C_1..C_k)} = \sum_k \sum_i \sum_j P(A_i=V_{ij}) * P(C_k \mid A_i=V_{ij}) * P(A_i=V_{ij} \mid C_k)$$

L'intégration proprement dite dans COBWEB consiste à modifier la liste des triplets du concept. La totalité des manipulations effectuées peut être comparée à une généralisation dans la mesure où la description du concept est "étendue" afin de "couvrir" le nouvel ensemble.

Finalement, il faut noter que la présence de plusieurs opérateurs signifie une recherche sur plusieurs directions dans l'espace des solutions. Pour cette raison la stratégie de contrôle de COBWEB a été définie comme un *hill climbing*.

3.3.2. ADECLU

Cet algorithme [Decaest 93] est très proche de COBWEB. Il utilise essentiellement le même schéma dans la construction. Les différences sont au niveau des notions manipulées.

Premièrement, pour chaque concept une distinction entre les couples attribut valeur est faite. Ceux-ci sont divisés en deux, des traits typiques pour le concepts et des traits présents mais non typiques. L'importance des premiers dans la construction des est prépondérante. Ce traitement sélectif est à la base de la notion de *concept statistique* dans ADECLU, par opposition aux concepts probabilistes. Ensuite, ADECLU utilise un critère d'évaluation différent. Sur ce point il fait un retour sur la notion de dissimilarité, en mesurant les correspondances partielles entre la description d'un objet et celle d'un concept. Finalement, cet algorithme élargit l'ensemble des opérateurs de modification définis dans COBWEB par des considérations plus subtiles.

3.3.3. Conclusions

Les méthodes de formation de concepts présentent l'avantage d'être incrémentales. Les techniques globales de construction qu'elles utilisent sont très proches de la *classification heuristique* [Clancey 84]. Par contre, la situation à un niveau de la hiérarchie rappelle ce que l'on a vu dans le cas précédent. Ainsi l'individu à intégrer est considéré par rapport aux classes d'une partition et son adjonction à chacune est évaluée selon un critère global. Ceci reprend essentiellement la démarche de CLUSTER.

Le rôle de la dissimilarité varie. Dans certaines méthodes elle a complètement disparu. Dans d'autres elle constitue, sous une forme cachée, le critère global de qualité qui guide la construction.

Ensuite, les descriptions des classes, les concepts probabilistes, sont maintenus différemment de la généralisation vue dans les deux cas précédents (KBG et CLUSTER). Lors de l'adjonction d'un individu, la description de la classe est "élargie" par trois opérations essentielles: élargissement de la liste (Attribut Valeur Probabilité) sous-jacente avec des nouveaux éléments, modification d'un certain nombre d'éléments existants et/ou exclusion d'un élément de cette liste.

3.4. Conclusions

Nous avons montré quelques exemples de techniques existantes de construction de taxonomies. Notre étude a permis de mettre en évidence l'importance des notions comme les

noyaux, la dissimilarité et la généralisation. Ici nous allons faire une synthèse sur ces trois notions.

La notion de noyau est déterminée par son rôle dans l'initialisation d'une classe et, souvent, de sa description. Les noyaux sont les individus qui influencent de façon décisive la construction des classes. Deux interprétations pour cette notion sont possibles. La première relève de la théorie des prototypes. Selon elle, le noyau serait le prototype d'un concept à découvrir [Esposito 94], il est déduit de la connaissance que l'on dispose sur le domaine. La deuxième est purement pragmatique: le noyau n'est que l'individu par lequel la construction de la classe commence. Les autres individus de la classe sont ensuite regroupés autour. Mais au fil de la construction le noyau peut se retrouver loin du vrai centre. La notion de noyau peut être explicite comme dans CLUSTER et ULS, ou implicite comme dans les techniques de type *plus proche voisin* où l'initialisation d'une classe est faite par un couple d'individus. On peut également reconnaître des fonctions d'un noyau dans les premiers éléments traités par un procédé de formation de concepts. Ceux-ci provoquent la création d'une nouvelle classe et influencent de cette façon l'évolution ultérieure de la taxonomie. Dans le cas de la formation de concepts, comme dans le cas de CLUSTER, une migration des noyaux par rapport à la classe peut se produire.

La deuxième notion importante que nous pouvons noter est celle d'une fonction qui évalue les ressemblances entre éléments du domaine (ou plutôt leurs représentations). Cette fonction est souvent introduite sous forme d'une similarité ou dissimilarité, mais des formes plus exotiques, telle l'intégration dans un critère global, sont également rencontrées (ADECLU et ULS). Un point de distinction important réside dans la nature des éléments comparés à l'aide de la fonction. Dans certaines méthodes, elle est définie sur les couples d'individus. CLUSTER fait partie de ce groupe. Dans d'autres cas la fonction ne fait (presque) aucune distinction entre individus et classes. Ici on peut citer KBG. Une autre possibilité est de définir la fonction sur les couples (classe, individu) comme le font ADECLU et ULS. Pour une partie majoritaire des méthodes présentées, la fonction joue un rôle décisif pour la construction des classes. D'autres l'ont réduite à un rang secondaire, l'exemple typique est CLUSTER, où la dissimilarité ne sert que dans le choix des noyaux. Le cas extrême est constitué par COBWEB qui a ignoré complètement les comparaisons entre individus.

Enfin, nous allons nous arrêter sur la notion de généralisation. Cette opération est définie à l'origine sur les formules logiques et elle est liée à la subsomption (cf. [Buntine 88]). Ici, nous nous permettons de considérer une définition plus large. Sur un formalisme de représentation donné sera appelée *généralisation* l'opération décrite comme suit. Elle prend en argument un ensemble d'expressions, dont l'interprétation dans le domaine modélisé peut varier entre un individu singulier et un ensemble d'individus. Elle rend une autre expression dont l'interprétation englobe les interprétations des expressions de départ. La généralisation ainsi définie est clairement une opération d'abstraction. Ainsi, elle constitue un outil principal pour la construction des descriptions des classes en catégorisation.

Dans sa forme d'origine la généralisation est employée par les méthodes des deux premiers groupes ci-dessus (CLUSTER, ULS, KBG). Les éléments à généraliser sont les descriptions des individus et, souvent, celles de classes. La formation de concepts utilise une forme plus subtile de généralisation. Elle s'exprime dans les opérations de modification sur les *concepts probabilistes*. Conformément à la définition ci-dessus et abstraction faite de leur nature probabiliste, ces transformations ont pour effet final d'élargir l'ensemble d'individus du concept.

4. Conclusion

Nous clôturons un chapitre essentiellement bibliographique. Son objectif est de présenter les solutions possibles du problème de construction de taxonomies. Il y a eu en effet l'énumération d'un ensemble de critères de comparaison au début. Ensuite ces critères ont été référés dans l'étude de trois grands groupes de méthodes de construction. Tout au long de la présentation sur ces approches distinctes nous avons essayé de donner en résumé les principes fondamentaux ainsi que les outils typiques pour une approche ou une méthode concrète. Notre

étude n'a pas couvert la totalité des approches existantes. Par exemple, nous n'avons pas traité ici la classification probabiliste non-incrémentale [Cheese 90]. D'après les observation que nous avons faites au fur et à mesure de la présentation des différents groupes, nous pouvons tirer certaines conclusions sur la façon dont les SC et la construction des taxonomies doivent être couplés.

Dans un premier temps, notre approche, de par ses objectifs, se situe plutôt dans le courant de la catégorisation conceptuelle. Les procédures en classification automatique qui se limitent en un ensemble de classes décrites en extension, même si elles ont un certain intérêt, ne peuvent pas résoudre complètement le problème. Les SC ne considèrent explicitement, il est vrai, aucune description pour les catégories. Mais celle-ci est tout de même présente bien que sous une autre forme. En effet, pour une catégorie dans un SC produit, les catégories projections sur les membres du produit jouent le rôle de restrictions sur les langages L_I sous-jacents. En prenant leurs interprétations réelles on obtiendrait une description valide de la catégorie dans le produit. De l'autre côté, nous cherchons à définir des procédés formels qui mettent en jeu des notions mathématiques telle la dissimilarité (cf. plus loin). Par ce fait nous nous situons dans la rangée de l'apprentissage symbolique-numérique.

Dans ce chapitre, nous avons décrit quelques-unes des notions fondamentales mises en jeu par la catégorisation. La (dis)similarité, la généralisation, la couverture sont des notions que l'on rencontre invariablement dans les différentes méthodes. L'intégration d'un procédé de construction de taxonomies au sein des SC demande donc que ces notions soient intégrées au sein du modèle. Par conséquent, nous avons dressé une liste de primitives dont la traduction vers les SC devrait précéder les travaux sur une méthode concrète. A la tête de cette liste nous avons rangé la définition d'une mesure de dissimilarité. Son rôle principal dans la construction des classes en fait une obligation pour nous. Il faut noter tout de suite que la notion de similitude dans les individus est *a priori* étrangère aux SC. Comme nous l'avons dit, celles-ci ne considèrent aucune description explicite pour les individus, pas plus pour les catégories. Ceci dit, l'introduction d'une telle fonction représente une extension du modèle. Une partie majoritaire du chapitre suivant est consacrée à la description de cette extension.

Contrairement à la dissimilarité, les deux autres notions dans notre liste, la généralisation et la couverture, trouveront une traduction quasi directe vers les SC. Ceci est dû au fait qu'elles sont basées sur une relation d'ordre entre formules logiques très proche de la sous-catégorisation en SC. Leurs définitions formelles en termes de SC font également l'objet d'une section dans le chapitre suivant.

Reste la question du choix de la méthode qui mettrait en marche toutes ces notions. Nous avons voulu intégrer dans un premier temps une méthode déjà existante. Malheureusement, aucune n'est universellement adaptée au contexte des systèmes classificatoires. Chacune des méthodes présentées s'appuie sur un formalisme concret et un ou plusieurs types de données particuliers. Notre choix devrait donc nous conduire vers un travail d'adaptation de la méthode sélectionnée. Le coût estimé d'une éventuelle adaptation a constitué un premier critère pour ce choix. Un autre critère était la forme des taxonomies dans TROPES auquel l'algorithme effectif devrait être intégré. Ce critère permet d'écarter les méthodes qui produisent des recouvrements à la place de partitions. Ceci s'applique, en particulier, à la démarche de KBG, qui est sinon *a priori* admissible dans le modèle des SC, d'autant plus qu'elle possède une certaine généricité sur les types de données. Ainsi, nous avons retenu les deux autres approches, parmi lesquelles nous avons tranché pour celle de CLUSTER. Ce choix a été justifié par les caractéristiques suivantes. Premièrement, l'algorithme montre une bonne division entre les parties génériques du traitement, c'est-à-dire celles qui ne distinguent pas les types différents, et les parties spécifiques. Ceci promettait une adaptation vers les SC à faible coût. Ensuite, les opérations effectuées lors de la construction des descriptions dans CLUSTER sont très proches du produit dans le modèle des SC. Quant aux méthodes de formation de concepts, malgré leur fort atout, à savoir l'incrémentalité, leur adaptation aurait demandé la traduction de leur appareil probabiliste lourd. C'est essentiellement pour cette raison que nous avons écarté ce choix. Dans l'intégration de la méthode CLUSTER (Chapitre IV) nous avons pu bénéficier des primitives décrites ci-dessus.

Chapitre III. Une mesure de dissimilarité pour les systèmes classificatoires

Dans la partie conclusive du chapitre précédent nous avons souligné le rôle important de la dissimilarité dans le cadre d'un algorithme de construction de taxonomies. Il est clair que cette notion manque dans le modèle de base des systèmes classificatoires. Afin de remédier à cette insuffisance, nous avons développé la définition d'une nouvelle mesure. Le présent chapitre est consacré entièrement à cette mesure que nous avons appelée *la dissimilarité topologique*. Nous commençons par la donnée de quelques principes de base pour la notion de dissimilarité (section 1). Ensuite, (section 2) nous énumérons certaines des mesures les plus souvent utilisées. D'un côté on y présente les mesures employées en analyse de données, de l'autre quelques mesures adaptées au formalisme des objets sont données. Dans la suite du chapitre nous exposons nos propres idées sur ce qui devrait être la dissimilarité sur un systèmes classificatoires. Ainsi, avec la section 3 nous proposons une image générale de la nouvelle mesure. La section suivante présente les résultats d'une étude sur le comportement de la mesure sur certains structures de domaines habituels. Plus loin, nous donnons la définition des deux autres notions, la généralisation et la couverture en termes de SC. A la fin de ce chapitre nous envisageons une expérimentation sur les types construites qui consiste à prendre différentes structures de SC, en choisissant des opérateurs de généralisation différents. La présentation détaillée de ce travail se trouve dans l'annexe B.

Pour plus de lisibilité toutes les preuves des propriétés énoncées dans ce chapitre sont renvoyées en annexe.

1. Principes

Dans cette section nous présentons la notion de dissimilarité telle qu'elle et perçue dans l'analyse de données. Une dissimilarité est une fonction réelle définie pour un certain type d'individus et possède des propriétés mathématiques obligatoires. Une définition formelle de la dissimilarité, tirée de [Diday+ 82] est proposée ci-dessous.

Définition 10

Soit un ensemble d'entités $D = \{d_i\}$, appelé *domaine*. Une fonction $\delta : D \times D \rightarrow R^+$ est une *mesure (indice) de dissimilarité*. si elle vérifie:

i) $\forall$ i, j $\delta(d_i,d_j) \geq 0$ *(positivité)*

ii) $\forall$ i, j $\delta(d_i,d_j) = \delta(d_j,d_i)$ *(symétrie)*

iii) $\forall$ i, j $\delta(d_i,d_j) = 0 \Leftrightarrow d_i = d_j$ *(réflexivité)*.

Les trois propriétés ci-dessus forment une sorte de base obligatoire. Il est vrai que dans certains cas particuliers on utilise des mesures qui ne les vérifient pas toutes les trois à la fois. Mais en général la tendance est d'aller plus dans la direction opposée, en démontrant d'autres

propriétés, qui viennent s'ajouter à cette base. Ainsi, une notion très couramment utilisée est celle de la *distance*.

Définition 11

Une fonction de dissimilarité δ qui possède la propriété:

iv) $\forall$ i, j, k $\delta(d_i,d_j) \leq \delta(d_i,d_k) + \delta(d_k,d_j)$ *(inégalité triangulaire)*
est appelée une *distance*.

Les fonctions de *distance* forment une sous-classe propre pour les dissimilarités. Celles-ci sont les dissimilarités les plus largement employées.

Si l'on va un peu plus loin dans les propriétés à inspiration géométrique, on retrouve la sous-classe des distances dites *arborées*.

Définition 12

Une *distance arborée* est une distance qui vérifie la propriété suivante:

v) $\forall$ i, j, k, l $(\delta(d_i,d_j) + \delta(d_k,d_l)) \leq \max((\delta(d_i,d_k) + \delta(d_j,d_l)) , (\delta(d_i,d_l) + \delta(d_j,d_k)))$ appelée *inégalité quadrangulaire*.

Comme il est indiqué dans [Gascuel 95], la propriété **iv**) est une conséquence de **v**). En effet, il n'y a pas de contrainte d'unicité pour les quatre éléments que l'on y retrouve. L'inégalité du triangle peut en être déduite dans le cas où il y a un couple d'éléments égaux. Une propriété intéressante des distances arborées est qu'une telle distance définit une structure d'arbre sur le domaine D. Plus précisément, il est possible de construire un arbre dont: *primo*, les feuilles correspondent aux éléments de D; *secundo*, les arêtes sont étiquetées par des nombres réels et les sommes des étiquettes sur un chemin entre deux feuilles équivalent à la valeur de la distance entre les éléments correspondants. Inversement, à partir d'un tel arbre on peut construire une distance arborée.

Finalement, une sorte de dissimilarité fortement liée à la structure de hiérarchie est la distance *ultramétrique*.

Définition 13

Une dissimilarité δ est une *ultramétrique* dans le cas où:

vi) $\forall$ i, j, k $\delta(d_i,d_j) \leq \max (\delta(d_i,d_k), \delta(d_k,d_j))$ *(isocellité du triangle)*.

Ceci n'est qu'une énumération partielle des mesures de différentes sortes de dissimilarité. Une classification exhaustive de ces fonctions se trouve dans [vCutsem 94].

Il faut noter que les dissimilarités sur un domaine se divisent en *observées* et *calculées* [vCutsem 94]. Les premières sont directement relevées sur les individus. Par exemple le poids pour un échantillon de la population, les distances entre des points sur une carte, etc.

Les dissimilarités calculées ne peuvent être définies que dans les cas où les individus dans D sont décrits par un ensemble de variables. Chaque variable représente une fonction v de D dans un autre domaine D_v. Généralement, les domaines D_v sont des domaines numériques ou ordinaux mais des domaines nominaux sont également considérés. L'essentiel est que chacun de ces domaines dispose d'une façon directe d'évaluer les proximités entre ses valeurs. En d'autres termes, on suppose un tel domaine D_v muni d'une fonction $\delta_v: D_v \times D_v \rightarrow R^+$.

Nous allons appeler les δ_v *fonctions d'écart*. En connaissant ces fonctions on définit la dissimilarité δ sur *D* comme suit.

Définition 14

Soient donnés le domaine *D* des individus à considérer et les domaines D_i correspondant aux variables v_i , ($i = 1,m$). Soient encore les fonctions d'écart δ_i sur les domaines D_i. Pour deux individus quelconques d et d' dans *D* la fonction de dissimilarité δ est calculée comme

$$\delta(d,d') = \boldsymbol{f}(\delta_1(v_1(d),v_1(d')),..., \delta_m(v_m(d),v_m(d')))$$

où $v_i(d)$ est la valeur de la variable v_i pour l'individu d et $\boldsymbol{f}$ est un opérateur complexe.

Il est clair que les propriétés de la fonction δ dépendent de celles des fonctions δ_i ainsi que de la nature de l'opérateur $\boldsymbol{f}$. Par exemple, si les δ_i sont des distances et $\boldsymbol{f} = \Sigma$, alors δ est aussi une distance (d'après [vCutsem 94]).

Une liste d'opérateurs réels pour $\boldsymbol{f}$ est proposée dans le paragraphe suivant.

2. Mesures existantes

Dans ce paragraphe nous allons citer quelques mesures de dissimilarité pratiques. D'abord, nous présentons les mesures en provenance de l'analyse de données dans le § 2.1. Le paragraphe suivant sera consacré aux mesures utilisées dans divers cas en apprentissage.

2.1. Mesures numériques

Dans ce paragraphe nous utiliserons les notations préalablement établies sur le domaine $D = \{d_i\}$. Rappelons que les variables v_i sont toutes à valeurs dans l'ensemble des réels. Les fonctions d'écart δ_i correspondent donc à la différence absolue sur les réels. Autrement dit

$$\delta_i(v_i(d),v_i(d')) = |v_i(d) - v_i(d')|.$$

Un premier groupe de mesures est formé par les distances dites euclidiennes (ou quadratiques).

Définition 15

Une distance δ est dite *euclidienne* s'il existe une matrice $W= \{w_{ij}\}$, symétrique et positivement définie, telle que:

$$\delta^2(d,d') = \sum_{i=1}^{m} \sum_{j=1}^{m} w_{ij} * |v_i(d) - v_i(d')| * |v_j(d) - v_j(d')|$$

Un exemple d'une telle mesure est la distance *euclidienne classique*. Dans ce cas la matrice $W = E$. On en obtient donc

$$\delta(d,d') = (\sum_{i=1}^{m} |v_i(d) - v_i(d')|^{2})^{1/2}$$

La distance de *Mahalanobis* est une variante de la précédante où les termes de la somme sont pondérés sur l'inverse de la variance correspondante. Autrement dit *W* est une matrice diagonale dont $w_{ii} = V_i^{-1}$.

$$\delta(d,d') = \left(\sum_{i=1}^{m} V_i^{-1} * |v_i(d) - v_i(d')|^{2} \right)^{1/2}$$

La distance de *Minkowski* définit une famille de distances. En fonction du paramètre λ on en obtient différentes fonctions dont une sous-famille stricte est constituée par les distances euclidiennes. Pour cette raison la *distance de Minkowski* est qualifiée de *semi-euclidienne*

$$\delta(d,d') = \left(\sum_{i=1}^{m} |v_i(d) - v_i(d')|^{\lambda} \right)^{1/\lambda}$$

La distance euclidienne présentée en premier est clairement un cas particulier de la mesure ci-dessus, avec $\lambda = 2$. Deux autres membres notables de la famille sont: la distance dite de *Manhattan* ou *"City block"*

$$\delta(d,d') = \sum_{i=1}^{m} |v_i(d) - v_i(d')|$$

avec $\lambda = 1$ et la distance de *Chebichev*

$$\delta(d,d') = \max_{i=1,m} |v_i(d) - v_i(d')|$$

avec $\lambda = \infty$.

2.2. Mesures symboliques

Les mesures de similarité ou dissimilarité utilisées dans le contexte symbolique se caractérisent par une hétérogénéité dans les données prises en compte. Ce fait explique l'existence de plusieurs cas traités à part [Bisson 95], [Gowda+ 94]. En effet, ce sont les fonctions d'écart qui changent pour les différents types de données. Par conséquent une telle mesure est de loin supérieure à celles des mesures cités préalablement. Précisons que notre but n'est pas de répertorier les mesures existantes mais de donner une idée sur leur fonctionnement. Pour cette raison nous allons décrire brièvement leurs principes.

Distance syntaxique

La *distance syntaxique* (angl. *syntactical distance*) [Mich+ 84] est historiquement la première parmi les mesures présentées ici. La mesure globale est une fonction définie sur les objets, décrits en APC (cf. Annexe). Elle représente une simple somme des fonctions d'écart sur les attributs. Celles-ci sont définies de façon propre à chaque type des données distingué: attributs *linéaires* (totalement ordonnés), attributs *nominaux* (sans ordre) ou attributs *structurés* (partiellement ordonnés). Pour les nominaux on utilise le complémentaire de la fonction identité, tandis que l'écart sur les linéaires est mesuré par la différence arithmétique. Les structurés ne bénéficient d'aucun traitement spécial, on leur applique l'une des fonctions citées plus haut selon l'existence d'un ordre "secondaire" sur leurs valeurs.

Dissimilarité dans ULS

La dissimilarité utilisée dans le système ULS [Espos 94] cité dans le chapitre précédent a été complètement définie dans [Espos+ 92]. Elle partage le langage des descriptions avec la dissimilarité syntaxique. Mais contrairement à cette dernière, la nouvelle mesure est définie sur les descriptions des objets et celles des classes. En effet, elle évalue la probabilité d'appariement, on parle d'un appariement partiel, entre un couple de descriptions (objet,

classe). Par conséquent c'est une mesure non symétrique. Elle distingue les mêmes trois types de données que précédemment. De plus, cette mesure est capable de prendre en compte la structure des objets composites.

Similarité dans KBG

Dans le cadre du système KBG [Bisson 92], une similarité entre formules en LPO est utilisée. De façon analogue à la mesure précédente, la similarité évalue un appariement partiel entre formules. Mais elle ne distingue pas les individus et les classes. Par ailleurs, beaucoup d'importance est attachée à la prise en compte de la structure dans la connaissance. Ainsi la similarité entre deux éléments, chacun pouvant être un individu ou une classe, est mesurée en fonction des éléments en relation avec ceux-ci. Et, puisque les relations circulaires sont autorisées, toutes les similarités sont évaluées à la fois par la résolution d'un système d'équations linéaires. Cette résolution passe par un appariement entre graphes et, dans une phase ultérieure, un calcul de point fixe. Une autre particularité à ne pas manquer est l'assignation de poids aux différents caractéristique des éléments.

Très récemment cette mesure a été redéfinie dans un cadre à objets [Bisson 95].

Similarité et Dissimilarité sur les Objets Symboliques

Les deux mesures définies dans [Gowda+ 94] ne sont pas propres à une méthode ou à un groupe de méthodes mais à un formalisme: les objets symboliques [Diday 93]. Rappelons que ceux-ci permettent un traitement en intension de analyse de données et qu'ils ne distinguent pas les individus des classes. Sur ce point il y a un accord entre les deux dernières mesures. Ensuite, ces mesures montrent un principe très original qui consiste à mesurer trois différents facteurs de ressemblance entre deux (ensembles de) valeurs, à savoir la cardinalité, la position mutuelle des valeurs et le contenu des valeurs. Comme dans tous les cas précédents on fait distinction entre différents types de données.

3. Dissimilarité topologique

Dans cette section nous allons donner la définition d'une mesure de dissimilarité que nous avons appelée *topologique*. Cette appellation reflète le principe fondateur de notre mesure: l'exploitation de la structure (topologique) d'un domaine. Sous structure topologique d'un domaine D, que cet abus du langage nous soit pardonné, nous comprenons ici la taxonomie de classes disponibles sur le domaine. L'idée nous a été suggérée par la définition d'un système classificatoire où toute l'information disponible sur les similitudes entre individus est "codée" dans la taxonomie.

Comment exploiter la structure de taxonomie? On peut facilement imaginer le graphe G dont les nœuds sont les catégories dans C et les arcs suivent la restriction transitive et symétrique de $\leq$. Nous proposons donc de se servir des chemins du graphe G, et plus précisément leur longueur (ou cardinalité si l'on veut car les arcs ne sont pas étiquetées) pour mesurer les proximités entre les individus.

Le lien étroit entre la dissimilarité et la taxonomie n'est pas notre découverte. En analyse de données, depuis longtemps on utilise les taxonomies comme une approximation et une compression d'une dissimilarité [Benkar 93]. Mais la démarche numérique s'arrête là. Nous voulons aller plus loin. Une fois la taxonomie construite, nous nous permettons de l'utiliser pour le calcul d'une fonction d'écart (de proximité) sur le domaine. La fonction va à son tour contribuer la dissimilarité sur un domaine d'individus composites. Une nouvelle taxonomie va être construite sur ce domaine en s'appuyant sur la dissimilarité. Cette deuxième taxonomie donnera naissance à une nouvelle fonction d'écart et ainsi de suite.

La définition stricte de la *dissimilarité topologique* en termes de systèmes classificatoires est proposée dans le § 3.1. Dans le paragraphe suivant nous décrivons la fonction d'écart qui contribue dans la dissimilarité. Ensuite nous étudions le comportement de la nouvelle mesure en fonction du domaine modélisé par le système classificatoire. Plus précisément, nous donnons

des résultats sur les propriétés de la fonction ainsi que sur la façon effective de calculer sa valeur.

3.1. Dissimilarité globale

Définir une dissimilarité opérationnelle dans le cadre des SC revient à préciser une fonction qui, pour un SC donné, plonge les couples d'individus dans l'ensemble des réels. En d'autres termes, on cherche une fonction d: $L_I \times L_I \rightarrow$ R qui vérifie les conditions d'une mesure de dissimilarité, éventuellement d'une distance. La fonction en question devrait être applicable sur un SC produit d'autres SC. Ensuite, elle doit faire apparaître comme seuls quantités variables les écarts mesurées sur les différentes dimensions du produit (soit δ_i, i=1..'nb dimensions'). Les fonctions d'écart doivent donc être identiques pour la totalité des SC membres. En ce qui concerne la fonction globale nous proposons qu'elle soit une somme des écarts sur les membres. La définition de la dissimilarité topologique peut être la suivante.

Soit $S = \langle C \leq \ll \rangle$ un système classificatoire (C et $\leq$ pouvant être vides) produit défini sur les langages L_C et L_I. Soient les systèmes classificatoires $S_1,...S_n$. ($S_j = \langle C_j \leq_j \ll_j \rangle$ et S_j est défini sur les langages respectifs L_I^j et L_C^j où $j \in$ [1 n]) tels que $S = \times_{j=1}^{n} S_j$ et les opérateurs de projection π_j ($j \in$ [1 n]) tels que $\pi_j(S) = S_j$. Soient encore e et $e' \in L_I$ deux individus quelconques dans S.

Définition 16

La dissimilarité topologique entre e et e' est définie par

$$d(e, e') = \Sigma_{j=1..n} \delta(\pi_j(e), \pi_j(e')),$$

où δ est la fonction d'écart entre les projections.

La façon exacte de calculer les fonctions d'écart est présentée dans le paragraphe suivant. Ici notons seulement qu'elles exploitent la structure de taxonomie sur chacun des membres et par conséquent ces systèmes classificatoires doivent être munis d'une telle structure. En d'autres termes C_j et $\leq_j$ ne peuvent pas être vides.

3.2. Fonction d'écart

Les fonctions d'écart sur les membres d'un système classificatoire sont la matérialisation du traitement générique discuté plus haut dans ce rapport. Ainsi elles partagent toutes la même définition, contrairement aux mesures du paragraphe 2.2. de ce chapitre. Rappelons que ceci leur permet d'être applicable à n'importe quel SC qui à son tour peut modéliser un type de données quelconque.

Le principe de la définition, nous l'avons dit, est de rendre compte de la position mutuelle des individus dans une taxonomie. C'est une façon de soumettre l'évaluation des ressemblances à la structure du domaine, exprimée à travers la taxonomie. Pour cela notre fonction d'écart va essentiellement dépendre du nombre des catégories *intermédiaires* pour deux individus. La notion de catégorie intermédiaire sera expliquée par un exemple. Prenons un système classificatoire $S = \langle C \leq \ll \rangle$ défini sur les langages L_C et L_I. Soit sa taxonomie $\langle C \leq \rangle$ comme celle dessinée sur Fig.19.

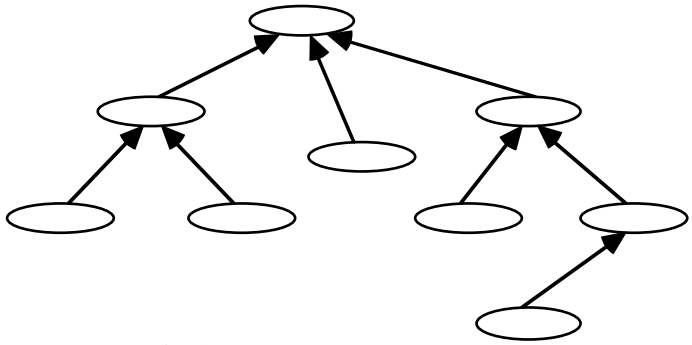


Fig.19. Un exemple de taxonomie.

Pour chaque individu e du langage L_I nous pouvons imaginer une catégorie $c \in L_C$ telle que $I_A(c)=\{e\}$. Maintenant, supposons que $c \in C$, sinon imaginons que la taxonomie est étendue par l'addition des toutes les catégories de ce genre. Pour ce faire il suffit de "raccrocher" la nouvelle catégorie sous la plus spécifique catégorie dans C qui contient (de par son interprétation) l'individu correspondant. Pour l'extension de l'ordre $\leq$ il suffit de faire ensuite sa fermeture transitive avec les nouveaux couples. La taxonomie obtenue peut être imaginée donc exactement comme le dessin sur Fig.20.

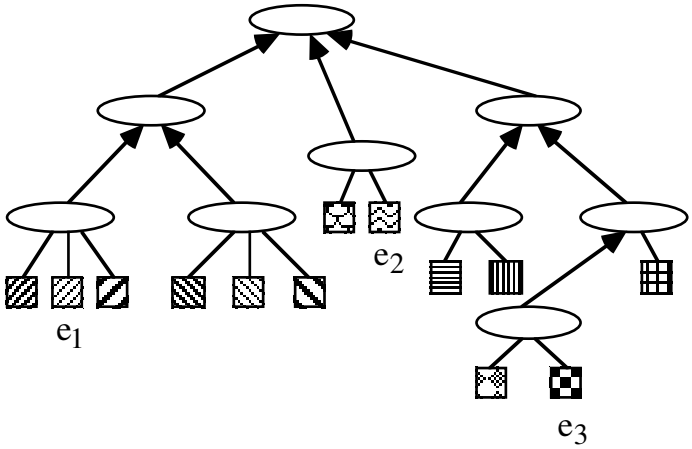


Fig.20. La taxonomie étendue par les catégories singletons.

Dans la suite nous faisons l'hypothèse que la taxonomie a une structure de demi-treillis supérieur. Plus loin nous allons montrer comment notre raisonnement peut être adapté au cas général.

La taxonomie étendue qui peut être vue comme un graphe orienté et, pour revenir sur la notion de catégorie intermédiaire, nous allons considérer les chemins dans ce graphe. Plus précisément, notre attention va se limiter aux chemins les plus courts entre deux catégories. Les nœuds internes d'un tel chemin seront essentiellement pour nous les catégories *intermédiaires*. Ensuite, pour définir les extrémités, considérons deux catégories singletons dans le graphe, par exemple e_2 et e_3 sur Fig. 20. Pour ce couple de catégories, il existe une unique sur-catégorie minimale dans la taxonomie. Nous considérons les chemins les plus courts qui vont de chacune des deux vers cette sur-catégorie. Sur le dessin celle-ci est la catégorie universelle. Dans le calcul de la fonction de l'écart les cardinalités des deux chemins sont prises en compte. Autrement dit, l'écart est fonction du nombre total des catégories intermédiaires entre les deux catégories singletons. Dans la suite nous allons confondre les catégories singletons aux individus correspondants.

Nous pouvons donner maintenant la définition de la fonction d'écart. Pour cela nous prenons les mêmes notations que dans le paragraphe précédent. Soit $S = \langle C \leq \ll \rangle$ un système

classificatoire défini sur les langages L_C et L_I. Soient e et $e' \in L_I$ deux individus quelconques dans S et $c \in L_C$ la sur-catégorie la plus spécifique dans la taxonomie étendue.

Définition 17

La fonction d'écart δ entre e et e' est définie par

$$\delta(e, e') = n + n',$$

où n et n' sont les cardinalités des chemins les plus courts entre e et e' d'une part et c d'autre part.

Pour prendre quelques exemples nous allons nous servir du dessin de la Fig.20. Ainsi, pour les trois individus indiqués les valeurs correspondantes de la fonction d'écart sont:

$\delta(e_1, e_2) = (3+2) = 5$

$\delta(e_1, e_3) = (3+4) = 7$

$\delta(e_2, e_3) = (2+4) = 6.$

Afin de garantir une égalité complète entre les différents membre d'un produit de SC, nous n'utilisons pas directement la fonction ainsi définie, mais une fonction normalisée. Pour cela nous effectuons une division sur la valeur maximale de la fonction d'écart, δ_{max}, sur la taxonomie sous-jacente. De cette façon les valeurs manipulées au niveau global sont toutes comprises dans $[0,1]$. La nouvelle fonction ainsi définie, δ_n est exprimée comme suit

$$\delta_n = \delta / \delta_{max}.$$

Maintenant, les valeurs effectivement calculées dans l'exemple ci-dessus sont

$\delta(e_1, e_2) = (3+2)/7 = 5/7$

$\delta(e_1, e_3) = (3+4)/7 = 1$

$\delta(e_2, e_3) = (2+4)/7 = 6/7.$

Nous pouvons maintenant revenir sur notre hypothèse de demi-treillis supérieure. Elle nous a donné la certitude que dans tous les cas une sur-catégorie unique existe. Elle n'est pas essentielle pour notre définition car en présence de plusieurs sur-catégories minimales, donc non comparables, nous pouvons encore choisir celle pour laquelle la somme des deux cardinalités est minimale. Et puisque ce dont nous nous servons n'est pas la catégorie mais les deux valeurs entières, le cas où plusieurs catégories correspondant à cela existent n'est pas gênant. Notre définition peut donc être modifiée afin de rendre compte du cas général.

Définition 17'

La fonction d'écart δ entre e et e' est définie par

$$\delta(e, e') = n + n',$$

où $n + n'$ est la moindre somme de cardinalités des chemins les plus courts entre e et e' d'une part et une sur-catégorie commune (minimale) d'autre part.

Nous pouvons maintenant nous interroger sur le type de la fonction que nous avons définie. Un premier résultat sera de montrer qu'elle représente un mesure de dissimilarité valide.

Théorème 1

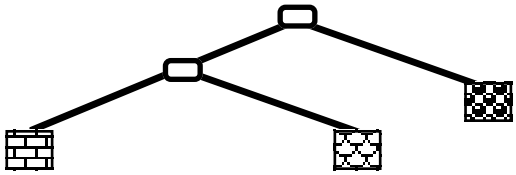
δ_n est un indice de dissimilarité.

Corollaire 1.1

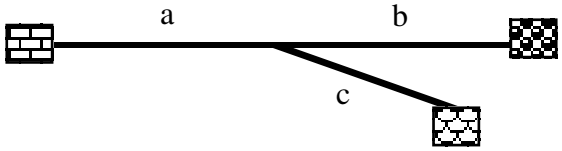
La dissimilarité topologique est aussi un indice de dissimilarité.

Une autre question que l'on peut se poser porte sur l'appartenance de la dissimilarité topologique à un autre groupe de fonctions plus restreint. Par exemple, nous pouvons chercher à vérifier dans quelles conditions elle est une distance. Avant de citer un résultat intéressant, nous allons faire une remarque utile pour la suite. Sans donner une preuve stricte (cf. [vCutsem 94]), nous affirmons le fait que l'opérateur d'addition sur les nombres préserve l'inégalité du triangle. Ainsi une condition suffisante pour que notre dissimilarité soit une distance est que toute fonction d'écart en soit une. Dans la suite nous donnons une propriété nécessaire et suffisante pour que la fonction d'écart soit une distance.

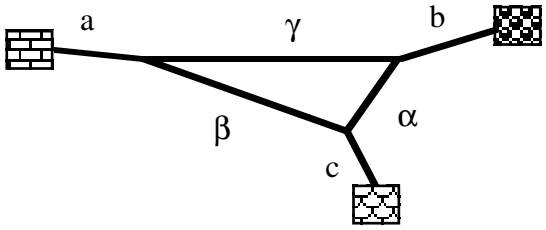
Tout d'abord signalons que nous cherchons une condition sur la structure de la taxonomie sous-jacente. On peut s'apercevoir qu'en cas d'une arborescence, la fonction d'écart est *a fortiori* une distance et même une distance arborée. En effet, prenons trois individus dans une telle taxonomie. La situation peut être schématisée ainsi.



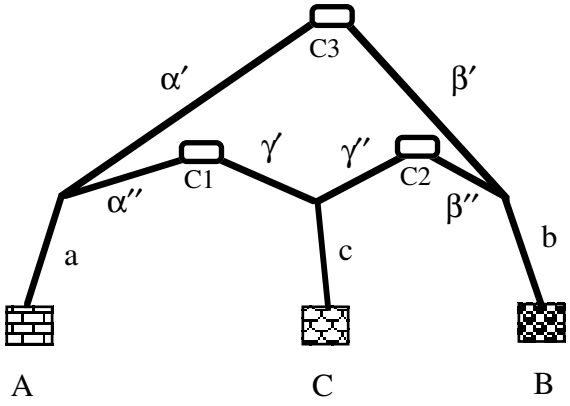
Ici nous faisons abstraction des catégories intermédiaires. Le fait important est que les chemins les plus courts sont en réalité les seuls chemins. Par conséquent, au moins deux des trois sur-catégories minimales sont identiques. En effet, nous pouvons schématiser encore plus la situation et la propriété d'une distance devient presque visible.



Dans le cas général, le problème est posé par le fait que les trois catégories sont distinctes. En d'autres termes, la situation se présente ainsi



Et si l'on veut indiquer les sur-catégories sur le schéma, celui-ci devient

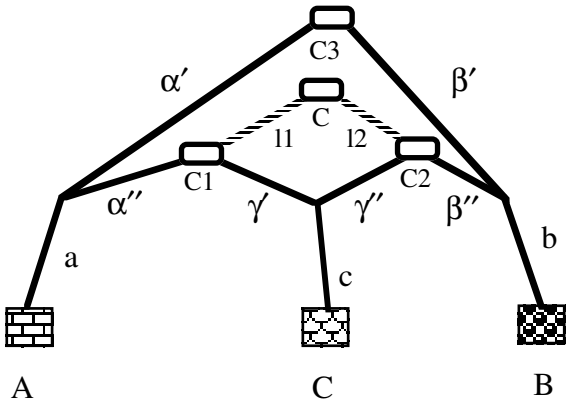


Les lettres indiquent les cardinalités des parties de chemins correspondantes. Elles peuvent être zéro dans le cas général. Afin de formuler une condition sur la structure de la taxonomie, nous devrons considérer ces valeurs et les lier dans une inégalité car le but est de garantir l'inégalité triangulaire. Sans perte de généralité nous pouvons considérer que les individus *A* et *B* sur le schéma ci-dessus montrent l'écart le plus élevé. Si l'on exprime ceci en cardinalités: $a + c + \alpha'' + \gamma' \leq a + b + \alpha' + \beta'$ et $b + c + \beta'' + \gamma'' \leq a + b + \alpha' + \beta'$.

Nous allons donc chercher une condition pour que $a + b + \alpha' + \beta' \leq a + c + \alpha'' + \gamma' + b + c + \beta'' + \gamma''$. Ceci revient à

$$\alpha' + \beta' \leq 2c + \alpha'' + \gamma' + \beta'' + \gamma'' \qquad (*).$$

Soit maintenant donnée *C*, la sur-catégorie de *C1* et *C2* minimale dont les chemins les plus courts à partir de ces deux catégories ont une longueur totale minimum. Soit encore les longueurs correspondantes de ces chemins *l1* et *l2*. La situation est la suivante



Nous savons que *C3* est la sur-catégorie à somme des chemins minimale pour *A* et *B*. L'inégalité suivante est donc valide $a + b + \alpha' + \beta' \leq a + b + \alpha'' + \beta'' + l1 + l2$. Après simplification on obtient

$$\alpha' + \beta' \leq \alpha'' + \beta'' + l1 + l2 \qquad (**).$$

Pour que l'inégalité (*) soit vérifiée une condition suffisante est que la partie gauche dans (**) soit inférieure à celle dans (*). En d'autres termes

$$\alpha'' + \beta'' + l1 + l2 \leq 2c + \alpha'' + \gamma' + \beta'' + \gamma''$$

et donc

$$l1 + l2 \leq 2c + \gamma' + \gamma''.$$

Cette condition a été appelée de *borne supérieure proche*. Elle est à interpréter ainsi: chaque fois que deux catégories *C1* et *C2* ont une sous-catégorie commune *C'*, pour tout individu *e* dans l'interprétation de *C'* la somme des cardinalités des chemins les plus courts de *e* vers *C1* et *C2* est supérieure à la somme des cardinalités des chemins les plus courts entre *C1* et

C2 et leur sur-catégorie la plus spécifique *C*. Nous pouvons formuler maintenant le résultat suivant.

Propriété 2

La fonction d'écart est une distance si pour toute paire de catégories *c* et *c'* dans *C* qui ont une sous-catégorie commune aussi dans *C* la condition de *borne supérieure proche* est vérifié.

La condition ci-dessus est vérifiée par la quasi totalité des SC que l'on manipule a ce jour dans TROPES (essentiellement des arborescences et des treillis complets). Il est possible d'imaginer tout de même une taxonomie qui viole cette condition. Un exemple en est donné sur la Fig.21.

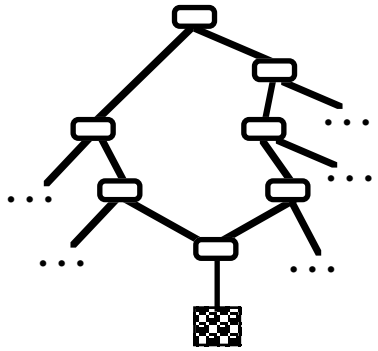


Fig.21. Une taxonomie qui ne vérifie pas la condition de *borne supérieure proche*.

4. Trois cas habituels

Dans le paragraphe précédent nous avons donné la définition d'une mesure de dissimilarité dite *topologique*. Son principe global est la mise à profit des structures taxonomiques sur l'ensemble des dimensions d'un SC produit. Nous avons également cité certains résultats concernant les propriétés universelles de la mesure.

Ici nous adoptons une approche pratique envers la nouvelle mesure. Les résultats cités ont de l'importance pour son utilisation effective. Notre objectif sera de montrer son comportement dans certains cas particuliers que l'on rencontre souvent dans la littérature. L'idée sous-jacente est, bien entendu, de montrer les avantages de la dissimilarité topologique par rapport aux mesures existantes.

Dans la suite, trois types de données: les domaines totalement ordonnés, non ordonnés ou énumérés, et partiellement ordonnés sont examinés.

4.1. Domaines énumérés

Les types énumérés sont typiquement utilisés pour modéliser des informations comme par exemple les noms des lieux, les marques des véhicules, etc. Aucune relation permettant de comparer les éléments d'un tel domaine n'est disponible *a priori*[3]. Il est clair qu'un concept n'est jamais un type énuméré par la présence d'une taxonomie de classes et l'ordre partiel sous-jacent. Les domaines énumérés correspondent donc en TROPES qu'à des ADT.

---

[3] Ceci n'empêche leur modélisation d'introduire de telles relations de manière indirecte.

En ce qui concerne la représentation d'un tel type sous forme de SC, celle-ci reproduit le treillis des ensembles sur ce domaine. En effet, la taxonomie pour l'interprétation abstraite n'a aucun raison d'exclure une possibilité ou une autre. Par conséquent, à chaque ensemble de valeurs du domaine correspond une catégorie du langage L_C.

De façon formelle, soit $S = \langle C \leq \ll \rangle$ un SC défini sur L_C et L_I qui modélise le type énuméré *T*. Soit aussi $D = E(T)$ le domaine de *T*. Pour la taxonomie abstraite nous avons

$$\langle L_C \ \ll \rangle \cong \langle 2^D \subseteq \rangle.$$

Or nous avons noté que pour un ADT les deux taxonomies, $\langle L_C \ll \rangle$ et $\langle C \leq \rangle$ sont identiques. Par conséquent, la fonction d'écart sera mesurée sur une structure isomorphe au treillis des sous-ensembles et nos considérations peuvent se rapporter sur celui-ci. En réalité nous serons amenés à considérer les chemins dans le graphe sous-jacent qui vont des ensembles singletons vers les ensembles doubles. Cette situation est illustrée sur la Fig.22. Les éléments du domaine sont exprimés par des lettres. Le dessin représente une partie du treillis avec deux ensembles singletons et l'ensemble double correspondant.

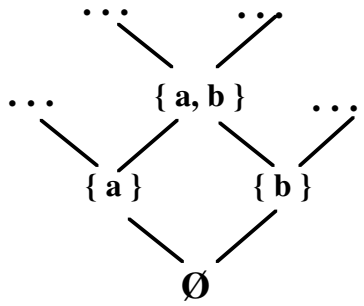


Fig.22. Une partie de taxonomie sur un domaine énuméré.

Dans ce contexte, la fonction d'écart va avoir deux valeurs possibles: *1* et *0*. En effet, tous les chemins calculés sont de longueur *1* ou *0* et pour deux éléments différents la somme des chemins est *2*. Après la normalisation nous obtenons pour deux individus *e* et *e'* dans L_I :

$\delta_n(e, e') = 0$ si $e = e'$ et

$\delta_n(e, e') = 1$ sinon.

Ceci constitue une façon pratique de calculer les valeurs de la fonction d'écart, sans recours aucun à la taxonomie. Il est clair qu'une telle technique sera à préférer pour une réalisation. Enfin, signalons que sur ce point notre mesure est en accord parfait avec les techniques existantes Diday+ 82], [Mich+ 84], [Espos+ 92].

4.2. Domaines totalement ordonnés

L'accord entre les fonctions d'écart usuelles et notre propre mesure peut être reconduit dans le cas d'un type totalement ordonné. Ce genre de types est typiquement utilisé pour les caractéristiques numériques et certaines autres dont le domaine est muni d'un ordre total. La fonction d'écart par excellence dans ces cas est la différence arithmétique. S'il s'agit d'un type non numérique, on fait recours aux nombres ordinaux des éléments. La normalisation est aussi un fait courant, on se sert de l'écart maximal. Dans ce paragraphe nous ne nous occupons que de types non denses, le phénomène d'un type dense n'étant pas présentable de façon fidèle par un SC. De plus, les types manipulés peuvent être infinis, mais à un instant donné nous ne considérons que des sous-ensembles finis.

Dans le modèle des SC une prise en compte directe des relations entre les individus n'est pas possible. Nous ne pouvons que nous baser sur la taxonomie du domaine. Par contre celle-ci peut rendre compte de façon indirecte de la présence d'un ordre total. Cette propriété relève d'une modélisation correcte du type en question: choix du langage de catégories L_C et du critère de sous-catégorisation $\ll$. Ceux-ci doivent être le langage des intervalles et l'inclusion entre intervalles respectivement. Si ces deux conditions sont respectées, alors notre mesure rejoint la différence (normalisée). Pour illustrer ce fait, prenons l'exemple d'un type entier dont

les valeurs sont comprises dans [*0,3*]. Le langage des classes pour modéliser ce type sera $\Im_{[0,3]}$, l'ensemble des intervalles sur [*0,3*] avec l'inclusion entre intervalles d'entiers pour $\ll$. La taxonomie correspondante est dessinée sur Fig.23.

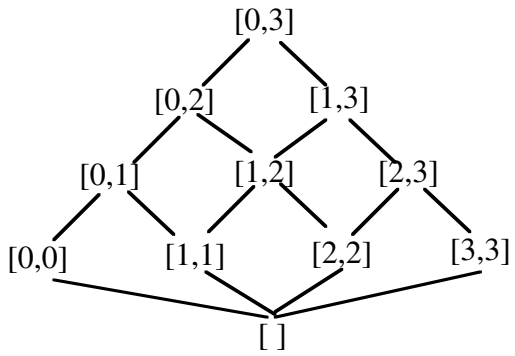


Fig.23. Un exemple de domaine totalement ordonné (entiers).

Il faut noter deux faits importants. Premièrement, pour deux individus quelconques *a* et *b*, si $a < b$, alors la catégorie minimale commune dans la taxonomie correspond à [*a*,*b*]. Ensuite, les chemins les plus courts dans le graphe montrent une propriété intéressante. Un tel chemin qui va d'une catégorie singleton (un intervalle à un élément) vers une sur-catégorie a pour cardinalité la différence des deux bornes de l'intervalle sous-jacent. Ainsi le chemin entre [*1*,*1*] et [*1*,*3*] a une cardinalité *2*, tandis que celui entre [*1*,*1*] et [*0*,*3*] - *3*. En effet on voit le mécanisme caché derrière ce phénomène: les intervalles forment des couches ordonnés pour la cardinalité. La combinaison entre ces deux faits permet d'obtenir les résultat final suivant.

Soit $S = \langle C \leq \ll \rangle$ un SC défini sur L_C et L_I et modélisant un type *T* totalement ordonné pour l'ordre $\leq_T$. Soient *e* et $e' \in L_I$ deux individus dont les interprétations respectives sont les valeurs *v* et $v' \in$ E(*T*).

Propriété 3

La fonction d'écart coïncide, dans ce cas, avec la différence absolue des ordinaux. En d'autres termes

$$\delta(e, e') = \text{abs}\ (\text{ord}(v) - \text{ord}(v')).$$

La fonction d'écart normalisée peut être calculée comme suit

$$\delta_n(e, e') = \frac{\text{abs}\ (\text{ord}(v) - \text{ord}(v'))}{\text{ord}(\max_e(T)) - \text{ord}(\min_e(T))}.$$

A noter que la deuxième formule fait apparaître les deux fonctions $\max_e$ et $\min_e$ qui ont la particularité de rendre non pas les extrêmes absolus d'un type, mais ceux effectivement référés dans un ensemble de SC. Cette technique nous permet de ne manipuler que des valeurs finies. Jusqu'à lors, il y a un accord entre les fonctions habituelles et notre mesure dans le cas d'un type totalement ordonné. Mais notre fonction d'écart présente l'avantage de pouvoir se généraliser sur les catégories, c'est-à-dire sur le langage L_C.

Pour cela nous définissons la fonction $\delta_\Im$ sur $\langle L_C \ll \rangle \cong \langle \Im \subseteq_\Im \rangle$ où $\Im$ est l'ensemble des tous les intervalles sur un type totalement ordonné et $\subseteq_\Im$ est l'inclusion entre intervalles. La structure $\langle \Im \subseteq_\Im \rangle$ est clairement un treillis, sous-treillis de $\langle 2^D \subseteq \rangle$.

La fonction $\delta_\Im$ possède toutes les caractéristiques de δ sur L_I. En particulier, $\delta_\Im$ hérite la façon dont les chemins dans le graphe sont considérés. Dans la suite nous allons prouver des

résultats sur cette nouvelle fonction dont les propriétés de δ sont des cas particuliers. Ainsi nous montrons qu'elle peut être calculée comme la cardinalité d'une *différence symétrique généralisée* sur les intervalles, notée $\Delta_{\Im}$. En d'autres termes pour I et J deux intervalles dans $\Im$, $\delta_{\Im}(I,J)$= card($I \Delta_{\Im} J$). Nous allons également montrer que cette fonction est une distance. Afin de simplifier les notations, tous les exemples seront désormais pris sur les entiers. Mais les résultats rapportés sont applicables à n'importe quel type totalement ordonné.

Notre premier but sera de définir la *différence symétrique généralisée*. Tout d'abord nous introduisons une notion analogique à l'*union ensembliste* par rapport à l'*inclusion* entre intervalles.

Définition 18

Soient $I = [a_1,b_1]$ et $J = [a_2,b_2]$ deux intervalles dans $\Im$. Nous allons appeler *l'enveloppe* de I et J l'intervalle $[\min(a_1,a_2),\max(b_1,b_2)]$. Cet intervalle sera noté $I \oplus J$.

L'enveloppe de deux intervalles peut être définie comme le plus petit intervalle les contenant. En prenant en compte le treillis $\langle \Im, \subseteq_{\Im} \rangle$ l'enveloppe correspond à la borne supérieure des deux intervalles. Il est clair que $\Im$ est fermé par l'opération $\oplus$. De plus, elle peut être généralisée sur un nombre quelconque d'intervalles dans $\Im$.

Définition 19

Avec la même hypothèse pour I et J, appelons la *différence sur enveloppe* entre I et J le résultat de la différence ensembliste $(I \oplus J) - J$. On note $I \otimes J$. La différence sur enveloppe entre J et I est définie de façon symétrique.

Selon la position respective de deux intervalles, l'ensemble résultant peut correspondre à un intervalle, vide ou non, ou bien deux intervalles. Quelques exemples sont donnés ci-dessous.

Ex. Soient $J = [2..4]$, $I = [7..11]$, $K = [8..9]$ et $L = [9..13]$. Nous en déduisons les enveloppes suivantes $I \oplus J = [2..11]$, $K \oplus J = [7..11]$, $K \oplus L = [8..13]$. Ensuite, les différences sur enveloppe deviennent:

$I \otimes J = [5..11]$,

$J \otimes I = [2..6]$,

$I \otimes K = [7..7]; [10..11]$,

$K \otimes I = []$,

$K \otimes L = [8..8]$,

$L \otimes K = [10..13]$,

Finalement, nous définissons la différence symétrique généralisée (sur enveloppe) de la façon suivante.

Définition 20

Avec la même hypothèse pour I et J, la *différence symétrique généralisée (sur*

enveloppe) entre I et J est le couple des deux différences sur enveloppe ($I \otimes J$) et ($I \otimes J$). On note $I \Delta_{\Im} J$.

Comme son homologue ensembliste, $\Delta_{\Im}$ est une notion symétrique. Son résultat est un ensemble d'intervalles. Nous ne nous intéressons plus loin qu'à une caractéristique très particulière de la différence symétrique généralisée, à savoir la cardinalité cumulée de ses éléments. A cette fin, le résultat suivant est d'utilité.

Propriété 4

Pour deux intervalles $I = [a_1,b_1]$ et $J = [a_2,b_2]$ il y a au plus deux intervalles non vides dans $I \Delta_{\Im} J$.

Maintenant nous allons revenir sur le calcul de la fonction $\delta_{\Im}$. Notre but sera de prouver que $\delta_{\Im}(I,J)= \text{card}(I \Delta_{\Im} J)$. Comme dans le paragraphe précédant, nous allons prouver une propriété plus précise.

Lemme 5

Soient $I = [a_1,b_1]$ et $J = [a_2,b_2]$ deux intervalles dans $\Im$. Considérons la longueur du plus court chemin dans le graphe du treillis $\langle \Im, \subseteq_{\Im} \rangle$ allant du sommet correspondant à I jusqu'à celui correspondant à $I \Delta_{\Im} J$. Cette longueur est égale à la cardinalité cumulée de $I \otimes J$.

Corollaire 5.1

Puisque $\text{card}(I \Delta_{\Im} J) = \text{card}(I \otimes J) + \text{card}(J \otimes I)$, et la longueur du chemin est additive, alors $\delta_{\Im}(I,J)= \text{card}(I \Delta_{\Im} J)$.

Ceci nous permet d'affirmer, que pour les intervalles $I = [a_1,b_1]$ et $J = [a_2,b_2]$, on a $\delta_{\Im}(I,J) = \text{card}([\min(a_1,a_2),a_1-1], [b_1+1,\max(b_1,b_2)], [\min(a_1,a_2),a_2-1], [b_2+1,\max(b_1,b_2)])$. En sachant que $\text{card}([a,b]) = b - a + 1$, nous pouvons conclure que:

$\delta_{\Im}(I,J) = a_1 - \min(a_1,a_2) + a_2 - \min(a_1,a_2) + \max(b_1,b_2) - b_1 + \max(b_1,b_2) - b_2$.

Ou encore:

$\delta_{\Im}(I,J) = a_1 + a_2 - 2*\min(a_1,a_2) + 2*\max(b_1,b_2) - b_1 - b_2$. (*)

Théorème 6

$\delta_{\Im}$ est une distance.

Enfin, signalons que même si nous avons laissé complètement à coté les types denses, comme les nombres rationnels ou réels par exemple, c'est parce que la technique de comptage

des catégories ne s'applique pas à ces types. Mais si l'on admet qu'un tel type puisse être représenté de façon exhaustive, c'est-à-dire que tous les intervalles (il y a un nombre infini) sont modélisés, alors un raisonnement très semblable peut être appliqué. D'abord, le fait que pour deux valeurs quelconques *a* et *b* telles que $a < b$, la catégorie commune la plus spécifique est l'intervalle [*a*,*b*] est toujours valide. Et si l'on admet que le nombre de valeurs dans l'intervalle est proportionnel à sa "longueur", c'est-à-dire à la différence $b - a$, alors nous pouvons remarquer le fait suivant. Plus un intervalle est "large", plus il est situé haut dans la taxonomie. Par conséquent, il est plus loin du niveau constitué par les intervalles singletons, tout en bas de la taxonomie (cf. Fig.23). Il existe donc une dépendance directe entre la position relative (sur l'axe vertical) d'un intervalle dans la taxonomie et la différence de ses bornes. Si l'on prend maintenant un couple d'individus, les chemins les plus courts jusqu'à la catégorie commune la plus spécifique sont proportionnels à la "longueur" de l'intervalle sous-jacent. Nous pouvons assumer que cette proportion est de coefficient constant (pas nécessairement égal à un). En prenant la forme normalisée, une fraction qui a la "longueur" de l'intervalle à mesurer pour numérateur et celle du domaine entier pour dénominateur, on en déduit donc la formule traditionnelle de la différence entre les deux nombres (réels).

4.3. Domaines partiellement ordonnés

On parle de types de données partiellement ordonnés dans le cas un domaine est muni d'un ordre partiel qui n'est pas total. Un exemple de type primitif partiellement ordonné est la caractéristique `forme géométrique` illustrée sur Fig.24. Ici l'ordre partiel est introduit par l'inclusion entre classes (au sens habituel du terme) de figures géométriques. Ainsi un `rectangle` est un `trapèze` et un `carré` est en même temps un `rectangle` et un `losange`. Autre type ou groupe de types partiellement ordonnées sont les concepts, l'ordre partiel dans ce cas est la spécialisation entre classes. Le moment important dans les deux cas, c'est l'inclusion des ensembles de valeurs qui est toujours présente derrière un tel ordre.

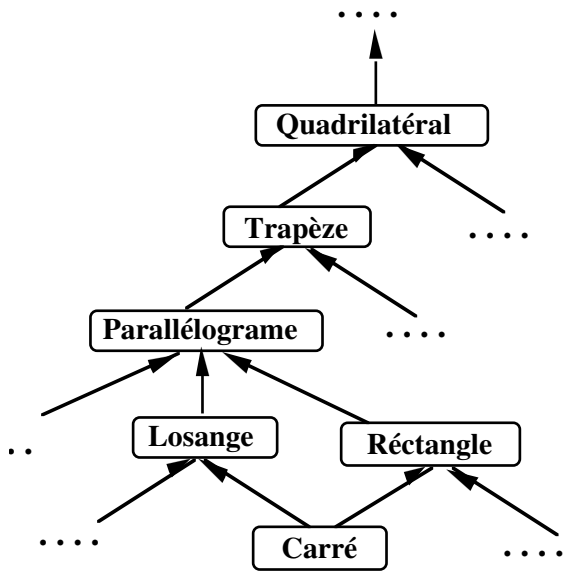


Fig.24 Un exemple de taxonomie sur le domaine des figures du plan.

Les domaines partiellement ordonnés ne sont pas universellement reconnus. Seuls certains approches en tiennent compte et ceci de façon incomplète [Mich+ 84], [Espos+ 92]. Le traitement qui leur est accordé ne tire pas tous les profits potentiels de la structure taxonomique sous-jacente. En particulier, celle-ci n'est pas exploitée par les mesures de similarité ou dissimilarité comme telle, même si des pas sont faites dans cette direction dans [Espos+ 92].

Une telle lacune dans les approches existantes nous prive de base pour la comparaison avec notre mesure. Nous ne citons donc pas d'exemple concret dans ce paragraphe, d'autant plus que l'exemple donné lors de la définition de notre fonction est un type partiellement ordonné.

Quand il s'agit d'un domaine d'objets notre fonction présente un autre avantage par rapport aux approches existantes. Celles-ci exploitent les objets de façon qui, dans sa finalité, revient à faire un aplatissement. Les objets du domaine sont considérés avec toutes leurs caractéristiques et si certaines sont des objets aussi le même principe leur est appliqué. Par conséquent, toutes les informations qui concernent, de près ou de loin, un objet sont prises en compte. Même si un tel principe est bénéfique pour la précision des comparaisons, la complexité derrière peut devenir pénalisante. Pour notre part nous nous tenons à un principe globalement valide pour tous les traitements de notre méthode. Celui-ci veut que lors de l'exploitation de la structure dans les objets, on ne descend pas plus bas qu'à un niveau. En particulier, les objets d'un domaine sont considérés en tant qu'éléments des classes dans la taxonomie et plus comme des collections de valeurs. C'est sur ce point fondamental que nous distinguons: nous tirons le profit maximal de la structure d'ordre partiel sur un domaine, tandis que les méthodes existantes ne son pas capables de le faire.

5. Généralisation et couverture

Après avoir étendu le modèle des SC avec la notion de dissimilarité, il reste à montrer comment les deux autres notions, l'opération de généralisation et la relation de couverture seront traduites. Elles constituent la base pour la construction des classes et leurs descriptions dans CLUSTER. Par exemple, la couverture permet d'établir l'appartenance d'un individu à une classe définie en intension. La couverture a été définie dans un cadre logique avant, une définition formelle peut être trouvée dans [Buntine 88]. Quant à la généralisation, elle est également un héritage de la logique, elle est étroitement liée à la notion θ-subsomption (cf. le même article). Cette dernière permet d'établir une relation de généralité entre deux formules en LPO. Le rôle de la généralisation dans ce contexte est, étant donné un ensemble de formules, de fournir une nouvelle formule, plus générale que celles de départ. Dans la grande majorité des cas la généralisation la plus spécifique est recherchée.

Sur ce point CLUSTER constitue une exception car la généralisation utilisée fait objet d'un compromis entre deux facteurs. D'un côté on veut des généralisations maximalement spécifiques, mais de l'autre on cherche à éviter que les descriptions deviennent trop complexes (par des longues listes de valeurs). Pour cela on ce permet d'"arrondir" les descriptions. Un seuil est introduit dans CLUSTER afin de guider l'opération d'arrondi. Celui-ci assure que la généralisation n'est pas trop grossière.

Dans notre traduction nous avons supprimé ce seuil. Il n'a plus de raisons d'être car dans le contexte des SC nous ne considérons pas de descriptions: les catégories à construire sont des produits d'autres catégories. Par conséquent, une généralisation ne peut avoir qu'une catégorie comme résultat.

5.1. Généralisation

Avant de donner la définition de l'opérateur de généralisation il importe de signaler sa place exacte dans la construction des catégories. En effet, cet opérateur sera employé au moment où une proto-catégorie est préalablement construite et ce que l'on veut est l'éteindre de façon à ce qu'elle englobe un nouvel individu. Pour ce faire nous disposons des opérateurs de projection qui nous permettent de "descendre" sur les éléments du produit. Précisons aussi que sur chacun des SC membres une taxonomies est disponible. Ainsi on est capable de décomposer la généralisation sur le produit. Une généralisation est effectuée sur chaque membre du produit. Elle se sert des projections, celle de l'individu et l'autre, de la catégorie, ainsi que de la taxonomie pour trouver et rendre la catégorie la plus spécifique pour l'ordre $\leq$ sous-jacent qui vérifie les deux conditions suivantes. Premièrement, elle est une sur-catégorie pour la catégorie en argument. Ensuite, l'individu appartient à son interprétation réelle. Ce principe permet de comparer la généralisation à une montée dans les taxonomies des SC membres.

Finalement, la catégorie sur le SC global est le produit des catégories fournies en résultat sur les éléments.

Afin de formaliser les idées ci-dessus, nous définissons les deux opérateurs de généralisation. Comme dans le cas de la dissimilarité, le premier porte sur le produit dont la taxonomie est en train d'être établie. Il fait apparaître également des opérateurs sur les différents éléments du produit.

Pour des raisons de complétude, nous définissons nos opérateurs non pas sur un couple (catégorie, individu), mais sur un couple d'ensembles de tels.

Soit $S = \langle \emptyset\ \emptyset <<\rangle$ un SC produit sur L_C et L_I. Soit encore $\{e_i\} \subseteq L_I$ un ensemble d'individus et $\{c_j\} \subseteq L_C$ un ensemble de catégories. Notons par π_k la projection unitaire de S sur le k-ème SC membre.

Définition 21

La généralisation des deux ensembles $\{e_i\}$ et $\{c_j\}$ est définie par l'opérateur Gen comme suit:

$$\text{Gen}(\{e_i\}, \{c_j\}) = \times_{k=1,n}\ \text{Gen}_k^*(\{\pi_k(e_i)\}, \{\pi_k(c_j)\})$$

Ici les opérateurs Gen_k^* sont les opérateurs de généralisation sur les membres du produit définis ci-dessous.

Définition 22

La généralisation d'un ensemble d'individus $\{e_i\}$ et un ensemble de catégories $\{c_j\}$ dans un SC $S = \langle C\ \leq <<\rangle$ est défini par l'opérateur Gen* comme suit:

$$\text{Gen}^*(\{e_i\}, \{c_j\}) = \min_{\leq} \{c \mid c \in C\ ,\ \forall\ i\ \ e_i \in I_A\ (c),\ \forall\ j\ c_j \leq c\}$$

Pour prendre un exemple, supposons que le SC $S = \langle C\ \leq <<\rangle$ modélise un ADT de type entier et dont le langage des catégories est $L_C = \Im$. Soient à généraliser les deux ensembles $\{3, 6, 9\}$ des individus et $\{[4,7]\}$. Dans ce cas l'opérateur de généralisation Gen* donnera le résultat $[3,9]$. Autrement dit, Gen* $(\{3, 6, 9\}, \{[4,7]\}) = [3,9]$

5.2. Couverture

La relation de couverture est utilisée à un moment de la construction des catégories différent que celui de la généralisation. Ainsi, elle dispose déjà d'une catégorie sur le SC produit qu'elle n'a pas à modifier mais tout simplement à vérifier l'interprétation. Car la couverture d'un individu par une catégorie se traduit par l'appartenance du premier l'interprétation réelle de la catégorie. De ce fait, on pourra éviter la double définition à deux niveaux, sur le produit et sur chacun de ses membres, pour le prédicat de la couverture (Cover). La décomposition du calcul d'interprétation est, en effet, intégrée dans la définition du produit de SC. Ainsi nous pouvons nous limiter avec la définition suivante qui ne distingue pas les SC selon leur nature ou degré de complétion.

Soit $S = \langle C\ \leq <<\rangle$ un SC produit sur L_C et L_I (C et $\leq$ pouvant être $\emptyset$). Soient encore $e \in L_I$ un individu et $c \in L_C$ une catégorie.

Définition 23

La couverture de *e* par *c* est déterminée par le prédicat Cover comme suit

$$\text{Cover}(c,e) = vrai \text{ ssi } e \in I_A(c).$$

Nous pouvons donner un exemple en nous servant du même système classificatoire que dans le paragraphe sur la généralisation. Si maintenant on cherche à établir la couverture entre (*e* =) 2 et (*c* =) [*3,9*], alors la réponse sera positive. En d'autres termes Cover ([*3,9*], *2*) = *vrai*

6. Types construits

En TROPES un certain nombre de types construits sont utilisés. Dans la définition d'un type construit il y a la donnée d'un type de base et d'un constructeur. Les valeurs du type construit ne sont pas atomiques, mais composées à partir d'une collection de valeurs de base. La nature de la composition est déterminée par le constructeur. A ce jour, nous n'en disposons que de deux: les ensembles et les listes. Mais l'intégration de la notion de constructeur abstrait est à l'étude.

Un certain nombre des propriétés (opérations et relations) génériques sont attachées aux constructeurs. Par conséquent, une partie des traitements des valeurs de ce genre est directement effectuée sur les collections et ne fait (presque) aucun recours au type de base. Le traitement des types construits est encore relativement mal étudié. En particulier, il n'y a pas de notions universellement acceptées de ce que peut être la généralisation pour un tel type ou comment calculer la dissimilarité de deux ensembles. Les considérations que nous avons retrouvées dans la littérature ne se réfèrent qu'à des situations particulières [Gowda+ 94], [Bisson 95] quand il s'agit de mesurer une dissimilarité entre valeurs construites.

Notre objectif sera de montrer de quel façon la dissimilarité topologique s'exprime sur un système classificatoire modélisant un type construit. Pour cela nous allons considérer de différentes modélisations possibles pour un tel type. En d'autres termes, notre stratégie consistera à prendre des variantes de langages de catégories, L_C, avec leur critères de sous-catégorisation, $\ll$. Ceux-ci seront donnés dans l'ordre de la précision croissante qui va de pair avec la complexité des SC sous-jacents. L'opérateur de généralisation, défini préalablement va jouer un rôle important dans notre démarche. Les taxonomies étant exclusivement de treillis, la généralisation à partir de deux individus nous indique la sur-catégorie utilisée pour calculer la fonction d'écart topologique. Ainsi nous pourrons donner une expression directe des écarts en fonction de certaines valeurs mesurées directement sur les individus. Nous allons démontrer, une fois cette forme de la fonction topologique exhibée, certaines propriétés concrètes.

Par souci de lisibilité nous avons évacué la suite de cette section vers l'Annexe B. Le lecteur est prié de bien vouloir s'y référer pour l'examen de divers types de modélisations. Les preuves des propriétés énoncées dans cet annexe se trouvent, comme précédemment, dans l'Annexe C.

7. Insertion dans TROPES

Les primitives que nous avons défini plus tôt dans ce chapitre ont été intégrées dans TROPES. Les deux primitives liées à la relation de sous-catégorisation, la généralisation et la couverture y sont déjà présentes. En effet, la couverture est une composante de l'opération de classification en RCO qui permet, entre autre, de déterminer si un objet est instance d'une classe. S'il s'agit de valeurs primitives, la couverture relève de calcul d'appartenance à un domaine. Cette dernière relation est à la charge du système METEO. Quant à la généralisation, elle se traduit dans TROPES par une montée dans les taxonomies, dans le cas d'un concept où les taxonomies sont hiérarchiques, ou par un calcul du plus petit majorant pour un ADT. Les opérations qui permettent ce passage vers des sur-catégories sont réalisées a ce jour.

La troisième notion importante dont il a été question ici, la dissimilarité, n'était pas disponible avant sous sa forme générique. Nous l'avons introduit dans la définition des ADT. Maintenant, c'est une primitive que l'utilisateur peut définir lui-même. S'il ne dispose pas d'une fonction adaptée à son type, la fonction d'écart topologique est utilisée par défaut.

La même situation se répète chez les concepts. La fonction d'écart étant disponible sur les ADT, elle est étendue vers une dissimilarité sur les concepts primitifs. Au moment où un concept est muni d'une taxonomie, celle ci donne lieu à une fonction d'écart topologique. Par conséquent tous les autres concepts qui dépendent de ce concept peuvent en faire usage.

8. Conclusion

Dans le chapitre courant nous avons défini une nouvelle mesure de dissimilarité. Cette mesure est, de par sa définition, parfaitement adaptée au modèle des SC. Son principe est l'exploration des structures taxonomiques sur les SC membres d'un produit. Ainsi la dissimilarité entre deux individus dans le produit est fonction de l'écart de leur projections sur les différents membres. La valeur des écarts dépend de la position mutuelle des individus-projections dans la taxonomie sous-jacente. Pour la mesure ainsi définie nous avons donné un ensemble de résultats: considérations sur sa validité ainsi que des conditions sur les formes de taxonomies pour qu'elles aient certaines propriétés.

Une fois la définition de la dissimilarité topologique donnée, nous avons étudié son comportement dans un certain nombre de cas pratiques. Un des avantages à souligner est que, au moins dans les cas considérés, notre dissimilarité peut être exprimée par une formule simple sur une ou plusieurs valeurs faciles à calculer sur les arguments. Un autre avantage est que dans la plupart des cas elle est une distance. Ces deux propriétés dépendent, pour un système classificatoire donné, du langage de catégories L_C.

En ce qui concerne l'utilisation de la dissimilarité topologique, l'utilisateur a le libre choix de l'ignorer en profit d'une mesure externe au système. Ceci est en vigueur sur les ADT dont la définition supporte l'intégration d'une mesure de dissimilarité. Ce que nous fournissons est la possibilité d'avoir, sur n'importe quel système classificatoire, qu'il soit d'un concept ou d'un ADT, une dissimilarité générique dont la seule exigence est la présence d'une taxonomie.

La dissimilarité topologique peut servir dans divers contextes. Une de ses utilisations possibles, au sein d'un algorithme de catégorisation conceptuelle, est discutée dans le chapitre suivant. La construction de taxonomies n'est pas la seule opération à tirer profit d'une mesure adaptée à la structure taxonomique. On pourra intégrer la nouvelle mesure dans d'autres modules ou d'autres systèmes qui exploitent les ressemblances entre individus (sous-entendu objets). Par exemple pour calculer une classification moins stricte que la classification traditionnelle à l'aide de la dissimilarité. Un exemple de telle classification peut être retrouvé dans [Gabriel 94].

Maintenant que les primitives nécessaires à un algorithme de classification ont été introduites au sein des systèmes classificatoires (et de TROPES) un tel algorithme va être présenté dans le cadre du système.

Chapitre IV. L'algorithme de catégorisation conceptuelle

Dans ce chapitre nous présentons notre proposition d'algorithme pour la construction de taxonomies dans le cadre de la représentation par objet. Le principe fondamental que l'algorithme suit est la mise à contribution des informations "codées" par les taxonomies sur les sous-structures des objets. Sa démarche consiste à commencer avec le traitement sur les concepts primitifs, ceux qui ne sont décrits qu'en termes d'ADT. Ces derniers sont toujours munis d'une structure taxonomique et la construction de la taxonomie sur un concept primitif peut s'y poser entièrement. Une fois les concepts de ce genre sont traités l'algorithme "remonte" d'un niveau de plus dans la structure des objets. Ce niveau est constitué par les concepts décrits exclusivement par des ADT et concepts primitifs. Les taxonomies correspondantes à ces concepts sont construites en faisant référence au niveau inférieur, mais jamais plus loin. C'est-à-dire que l'algorithme ne fait pas appel aux taxonomies des ADT décrivant un concept primitif si ils ne sont pas directement référés par le concept en cours de traitement. De cette façon ascendante l'algorithme arrive au niveau le plus haut, c'est-à-dire sur les concepts globaux.

Lors du traitement d'un concept particulier, l'algorithme se limite aux opérations définies dans le cadre des SC, le modèle abstrait décrit dans Chapitre I. Ainsi, il s'affranchit des considérations sur les types de données utilisées et reste sur un niveau général, ce qui lui permet de prendre en compte un ensemble quelconque d'attributs (à condition que ces derniers soit modélisés sous forme de SC).

Rappelons qu'à la suite d'une étude des diverses méthodes de construction (Chapitre II), nous avons choisi celle de CLUSTER [Mich+ 84] pour l'intégrer aux SC. Par rapport à notre algorithme global l'adaptation de cette méthode se situe au niveau partition d'une classe. La procédure de partition est donc une version améliorée de l'algorithme. Sa conception a demandé la traduction d'un ensemble de notions propres à CLUSTER vers le modèle des SC telles les sélecteurs, les complexes, etc. Ensuite, la procédure s'appuie sur les primitives définies dans le chapitre précédent. Par rapport à la version de base, elle fait un usage plus astucieux de la mesure de dissimilarité qui, étant définie sur les taxonomies, est plus adaptée à un rôle de guide pour la généralisation. D'autres modifications d'ordre inférieur ont également été apportées.

Dans la suite nous présentons l'algorithme d'inférence dans son intégralité. La présentation commence par une section où nous décrivons la version originale de l'algorithme CLUSTER qui finit par un paragraphe consacré aux modifications importantes que nous avons apporté à cette version. L'algorithme même est décrit dans la troisième section.

1. Algorithme CLUSTER

Ce paragraphe est consacré à la description de l'algorithme que nous avons choisi pour l'adaptation à TROPES. En effet, CLUSTER est une famille d'algorithmes de catégorisation conceptuelle: CLUSTER/PAF, CLUSTER/2, CLUSTER/S [Mich+ 1984], [Stepp+ 1986]. Dans la suite nous allons dénoter par CLUSTER l'algorithme CLUSTER/2.

Nous allons présenter les deux points importants de l'algorithme qui n'ont pas été introduits dans Chapitre II: le langage de représentation des données et le critère d'évaluation de la qualité des partitions. Ensuite nous allons présenter le procédé de partition d'un ensemble d'objets de CLUSTER.

1.1. Formalisme

Le formalisme de présentation utilisé est le calcul des prédicats annotés, essentiellement développé dans [Mich 83]. C'est une extension pratique de la LPO, proche des formalismes centrés à objet. L'APC est présenté en détail dans l'Annexe A.

Trois types d'attributs sont distingués, en fonction de la structure de leur domaine: *nominal*, à domaine sans aucune ordre, *linéaire*, à domaine à un ordre total et *structuré*, à domaine partiellement ordonné.

L'algorithme utilise une mesure de dissimilarité dite distance syntaxique. Les idées de base de cette mesure ont été présentées antérieurement dans le chapitre III. Nous rappelons que les seuls ordres sur les domaines des descripteurs traités sont l'ordre total et l'ordre trivial (sans ordre). Par conséquent les variables à domaine structuré sont réduites vers un des deux types ci-dessus par la distance syntaxique.

1.2. Critère LEF

Le critère d'évaluation de la qualité dans CLUSTER est un critère de type LEF (*Lexicographic Evaluation Functionnal*). Ce dernier est un critère complexe composé d'une liste de critères concrets.

Les critères élémentaires cités dans [Mich+ 1984] sont:

- *ajustement* entre les données et les classes: le rapport entre les cardinaux de l'extension et celui de l'intension d'une classe est mesuré,
- *simplicité* des descriptions: le nombre total des sélecteurs est à minimiser,
- *facilité* de la discrimination entre les classes:
- *réduction* des dimensions de l'espace des descripteurs.

L'avantage principal d'un critère de type LEF est son extensibilité. L'utilisateur est libre, pour une application particulière, de définir et intégrer ses propres critères dans la liste globale voire de remplacer un des critères préétablis. A titre d'exemple nous pouvons citer le critère mesurant une valeur globale dépendant de la dissimilarité que nous avons trouvé dans [Esposito 94]. Malheureusement, c'est un critère qui ne peut pas être appliqué aux SC.

1.3. Procédure de partition

Un pas itératif commence par les noyaux. Les descriptions les plus générales des classes associées sont produites par l'algorithme de STAR [Mich 83], un algorithme de discrimination combinatoire. A partir d'un noyau donné et le reste des noyaux cet algorithme construit un ensemble de descriptions de la future classe qui vérifient les deux propriétés suivantes. Premièrement, elle couvrent le noyau choisi et aucun des autres. Deuxièmement, elle sont les plus générales possibles. De telles descriptions sont construites pour chacune des classes. Ensuite ces descriptions sont améliorées globalement (les unes par rapport aux autres) et rendues disjointes. L'étape suivante est le calcul du critère FEL et la génération des nouveaux noyaux en fonction de sa valeur. L'algorithme s'arrête quand l'ensemble des noyaux devient stationnaire ou bien aucune amélioration de FEL n'est possible.

L'algorithme ainsi décrit construit une partition d'un ensemble d'individus. Afin d'obtenir une taxonomie il est englobé par un module de hiérarchisation. La tâche de ce dernier est d'appeler l'algorithme principal récursivement sur chacune des classes construites et ceci jusqu'à la satisfaction du critère d'arrêt.

Le schéma de l'algorithme CLUSTER est donné sur la Fig.25:

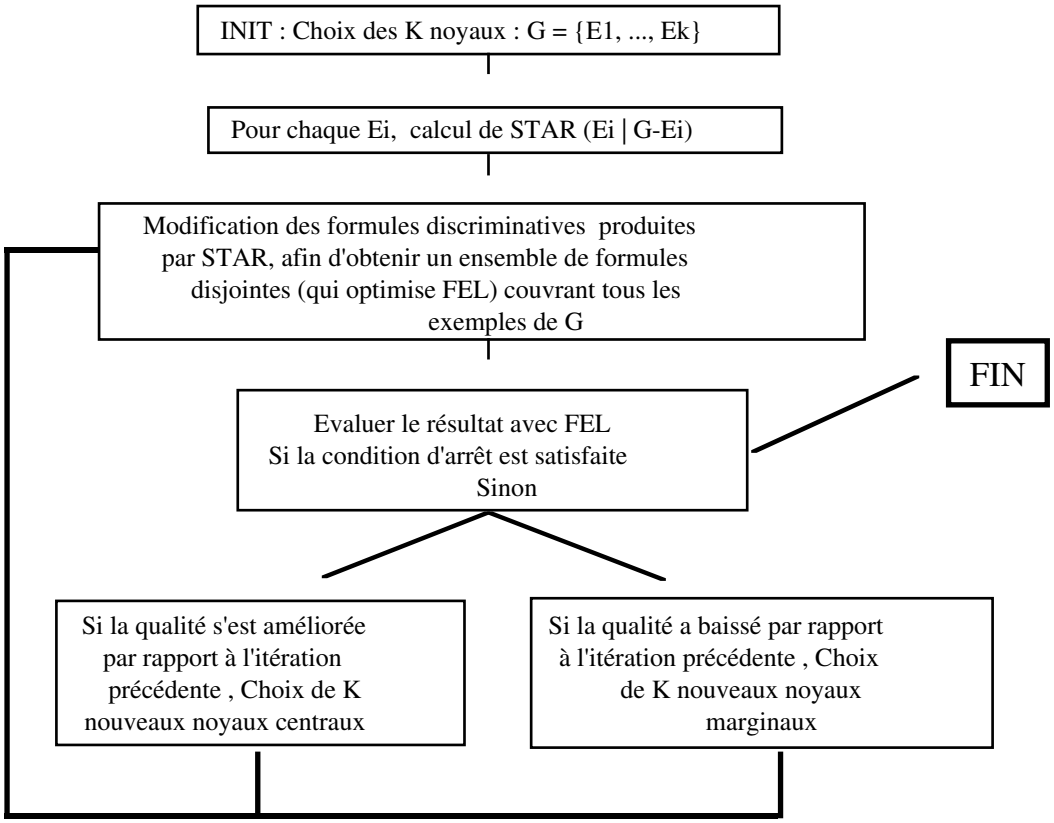


Fig. 25. Schéma de CLUSTER

1.4. Adaptation initiale pour TROPES

Sur la méthode présentée ci-dessus nous avons effectué les modifications essentielles suivantes.

Premièrement, le calcul des prédicats annotés a été remplacé par les SC. Ce remplacement a entraîné plusieurs reformulations, des notions utilisées par CLUSTER ont été traduites vers le langage des SC.

- *sélecteur* → catégorie sur un SC membre du produit
- *conjonction logique* → produit de catégories (sur les membres)
- *c-complexe* → produit de catégories sur les membres, dans le cas général, projection d'une catégorie sur le produit
- *d-complexe* → ensemble de (proto-) catégories

Une autre divergence entre les deux algorithmes est à observer au niveau de la mesure de dissimilarité. Notre algorithme utilise la *dissimilarité topologique*, définie dans le chapitre précédent. L'usage effectif de cette mesure constitue en soit un point de distinction. Dans notre algorithme la dissimilarité joue un rôle important lors de la distribution des individus sur les nuées des catégories. Plus précisément, nous l'utilisons pour trier les individus à distribuer. L'ordre est celui de la moindre dissimilarité avec un des noyaux des catégories. Après le tri, les individus sont distribués dans l'ordre. Ce traitement n'existe pas dans CLUSTER.

Ensuite, lors du traitement d'un élément dans cette liste, nous nous permettons de ne considérer qu'un nombre fixe de possibilités d'adjonction. Ces possibilités sont associées à la dissimilarité aussi car on ne compare que les adjonctions aux *r* catégories dont les noyaux sont les "plus similaires" (ou "moins dissimilaires").

Finalement, notre algorithme considère un certain nombre d'opérations comme atomiques. Dans cette liste on compte, entre autre, la généralisation et la couverture. Elles sont à la charge des deux systèmes, TROPES et METEO. Tandis que ces opération ont été explicitées dans CLUSTER, notre algorithme bénéficie d'une gestion qui est plus cohérente et plus efficace.

2. CLUSTER/SC

Dans cette section nous décrivons le prototype de l'algorithme CLUSTER/SC. Le fonctionnement de celui-ci peut être reparti sur trois niveaux chacun d'entre eux traité par un module à part.

Le niveau global est constitué par le traitement de la base de connaissances en entier. Une telle base définit un ensemble de systèmes classificatoires. La tâche effectuée par le module global est donc un parcours de cet ensemble. Ce parcours utilise la structure de graphe sans circuit induites sur l'ensemble de SC par la relation de décomposition entre un SC produit et les SC composants du produit. Le fonctionnement du module global est détaillé dans le paragraphe 2.1.

Pendant l'exploration du graphe décrit ci-dessus par le module global, un certain nombre de SC vont montrer une absence de taxonomie. L'objectif dans ce cas sera, bien entendu d'en construire une en s'appuyant sur les sous-SC. Cette construction constitue un deuxième niveau de l'algorithme qui se matérialise dans le module taxonomie. Le dernier procède par des partitions progressives des catégories traitées, en commençant par la catégorie universelle du SC. La manière exacte dont la construction se développe est discutée dans le paragraphe 2.2.

La partition d'une catégorie en sous-catégories est le troisième niveau du traitement. Le module qui en est chargé réalise une version révisée de CLUSTER. Il prend comme paramètre le nombre des catégories à construire et il rend une partition de bonne qualité. Dans le paragraphe 2.3. nous précisons la procédure de construction des sous-catégories.

2.1. Module global

Une base de connaissances à objet définit un ensemble de SC dont une partie sont des SC élémentaires et les autres - des SC produits. Si l'on considère la relation entre un SC produit et chacun de ses composants, cette relation, appelons la de *décomposition*, induit un ordre partiel strict sur l'ensemble de tout SC. La structure de graphe que l'on peut en extraire a été définie comme le *graphe des dépendances* (entre SC). Ce graphe joue un rôle important pour notre traitement au niveau global. Notons que celui-ci n'est qu'une notion abstraite, elle ne correspond à aucune structure dans l'algorithme ni dans sa réalisation.

Lors d'un appel du module global de notre algorithme, le graphe des dépendances est exploré en profondeur d'abord. L'exploration commence dans le noeud du SC à compléter par une taxonomie, passée en paramètre. A chaque pas la présence de la taxonomie réelle dans le SC du noeud courant est testée. Les SC dont une taxonomie est déjà présente ne reçoivent aucun autre traitement, ce sont les nœds terminaux de l'exploration. Dans ce cas se trouvent tous les SC qui modélisent les ADT de la base de connaissances ainsi que les SC de certains concepts préalablement structurés.

Sur le reste des nœds, l'exploration continue. Puisque dans ce second groupe ne se trouvent que des SC de concepts donc des produits, on appelle récursivement le module sur les nœds des SC composants.

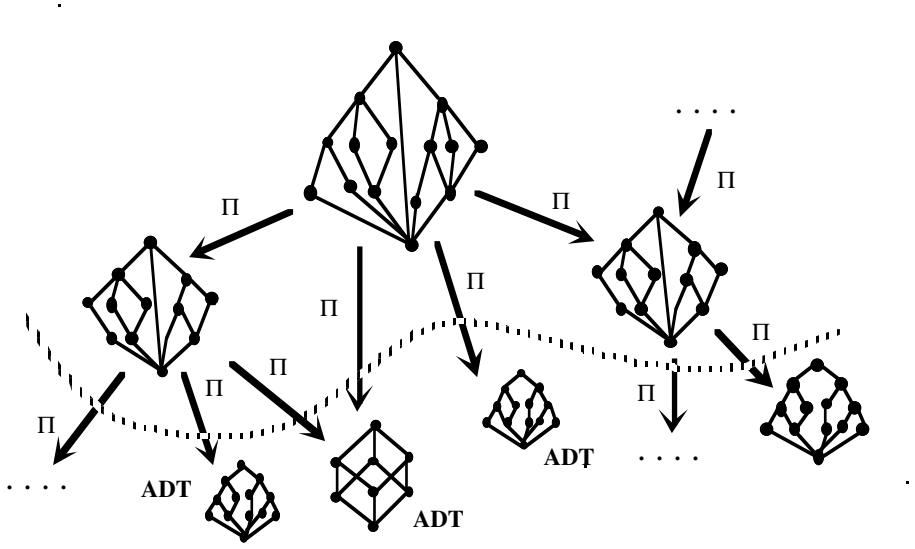


Fig.26. Une partie du graphe des dépendances avec l'indication de la ligne d'exploitation maximale.

Le traitement proprement dit n'est effectué qu'à la sortie de la récursion, le parcours global peut donc être caractérisé comme "post-fixe". Ainsi, une fois que l'algorithme a fini son travail sur les sous-SC, on est certain d'y retrouver une taxonomie réelle. Ceci permet de procéder à la construction de la taxonomie sur le SC courant. Pour cela on commence par la saisie d'un certain nombre de paramètres, avant d'appeler le module de construction de taxonomies.

Le schéma ci-dessus donne le pseudocode du module global.

---

procedure **CLUSTER/SC-Global** (
  $SC_gl = \{S_i\}$ - ensemble des systèmes classificatoires
  $ID_gl = \{E_i\}$ - ensemble d'ensembles d'individus
  ind - index du système classificatoire à traiter)

Algorithme:

if non (existe($<C_{ind} \leq_{ind}>$)) **then**

let $S_{ind} = \times_{j=1}^{n} S_j$

for j **from** 1 **to** n
CLUSTER/SC-Global(SC_gl , ID_gl, j)
endfor

SC = $\{S_j\}_{j=1,n}$
saisie_des_pametres(LEF)
CLUSTER/SC-Taxonomy(S_{ind}, E_i, SC, LEF)
endlet

endif

2.2. Module taxonomie

Le module de construction des taxonomies prend la main à un instant du travail de l'algorithme global où toutes les conditions pour la construction d'un taxonomie sont présentes.

En d'autres termes, on dispose d'un système classificatoire $S = <\emptyset \leq_{\emptyset} \ll>$ définit sur les langages L_I et L_C. Ensuite, un ensemble d'individus $E \subseteq L_I$ est donné. Le module reçoit également l'ensemble de tout sous-SC *SC*. La construction va être guidée par un critère de qualité lexicographique, LEF dont le principe a été expliqué plus haut.

Le fonctionnement du module s'articule autour d'une structure de données principale. Celle-ci est une file d'attente à protocole FIFO qui, à un moment donné, contient toutes les catégories à partitionner par l'algorithme. La squelette du module décrit une boucle sur toute catégorie dans la file. Au début la file d'attente est initialisée avec la catégorie universelle (ou top) de L_C. A chaque pas une catégorie sort de la file d'attente pour être partitionnée en sous-catégories. La partition est à la charge du module correspondant décrit dans le paragraphe suivant. Une fois cette partition effectuée, les nouvelles catégories obtenues sont aussi rentrées dans la file d'attente. Les seules exceptions à cette règle sont les catégories dont le nombre des individus dans *E*, ou leur interprétation réelle, reste au dessous d'un seuil. Celui-ci est fourni au module au début de son travail.

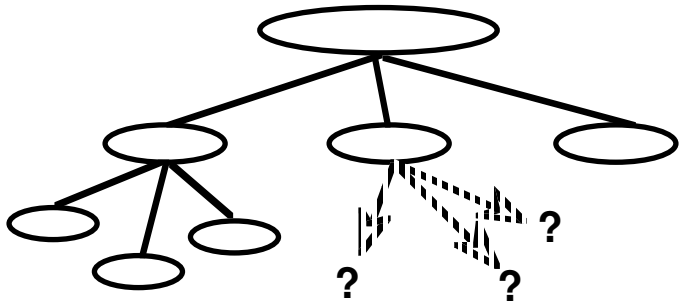


Fig.27. Le sens d'évolution de la taxonomie: la catégorie courante sera partitionnée.

Sur le squelette du module viennent se greffer d'autres traitements destinés à construire les éléments de *S*. Ainsi au fur et à mesure de l'évolution de la file d'attente on constitue l'ensemble des catégories *C* et la relation de sous-catégorisation ≤.

En dehors de la boucle principale reste un pas important de la construction, à savoir le calcul des dissimilarités entre les individus dans *E*. Celui-ci constitue une étape préliminaire dont le résultat est conservée dans une structure globale pour qu'il soit utilisé lors de chaque partition.

Le pseudocode du module taxonomie est fourni dans la suite. Nous avons présenté la variante qui n'exige pas la donnée du nombre de sous-catégories pour une partition. A la place, l'algorithme s'enrichit avec une boucle sur les cardinaux "raisonnables" d'une telle partition. Dans son article sur CLUSTER/2 Michalski "situe la raison" entre 2 et 7 [Mich+ 84]. A noter que la comparaison entre partitions de cardinaux différents demande plus de la flexibilité de la part du critère LEF.

---

procedure **CLUSTER/SC-Taxonomy** (

S	- système classificatoire à traiter
$E = \{id_i\}$	- ensemble d'individus de *S*
$SC = \{S_i\}$	- ensemble de sous-systèmes classificatoires
$LEF = \{c_q, t_q\}$	- le critère d'évaluation lexicographique)

Structures de données:

Ordre	- ensemble de couples de catégories, la réduction de l'ordre $\leq_{ind}$
Cour_Category	- liste de catégories
Best_Category	- liste de catégories
Queue	- FIFO de catégories

Algorithme:

 # Initialisation de la table des dissimilarités en utilisant δ , ***la dissimilarité topologique***
pré_traitement(*E* , *SC*)

Ordre = Ø
entrer(Queue, top(S_{ind}))
saisie_des_pametres(card_min)

while not vide(Queue)

  Best_Category = Ø
  curr_category = sortir(Queue)
  ens_courant = I_A(curr_category) ∩ *E*
  if card(ens_courant) ≥ *card_min* **then**
    saisie_des_pametres(b, p, card, r)
    for *k* **from** 2 **to** 7
      Cour_Category = **CLUSTER/SC-Partition** (ens_courant, *SC*, *k*, b, p, r, LEF)
      Best_Category = evaluer ({Cour_Category, Best_Category}, LEF)
    endfor
  endif

  for category **in** Best_Category
    C_{ind}= C_{ind} ∪ {category}
    Ordre = Ordre ∪ (category, curr_category)
    entrer(Queue, category)
  endfor

endwhile

2.3. Module partition

Le module partition est chargé de la construction, de manière itérative d'un ensemble de catégories. Son travail débute par le choix des noyaux initiaux (la procédure *initialiser_noyaux* dans le pseudocode ci-dessous). Celui-ci peut être fait de deux façons différentes. Dans une première variante les noyaux sont tirés au hasard. L'alternative consiste à prendre les individus qui maximisent la dissimilarité totale.

La suite du travail effectué par le module est encadrée par une boucle de contrôle. Cette boucle code une sorte de *hill climbing* dans l'espace des partitions en *k* catégories. Les pas élémentaires, sont évalués selon le critère LEF (fourni en paramètre), on ne garde que la partition qui donne le meilleur résultat. Le passage au pas suivant se fait par le choix des nouveaux noyaux (la procédure *calculer_noyaux*). Pour chacune des catégories construites le nouveau noyau est choisi parmi les individus dans son interprétation réelle. Les deux possibilités de choix sont les individus centraux, c'est-à-dire ceux qui minimisent la dissimilarité totale des catégories, ou les individus marginaux, ceux qui la maximisent. Ainsi à partir d'une partition on peut passer vers deux autres ceci fait deux directions pour le *hill climbing*.

Un pas de la boucle est divisé en deux grandes étapes: la formation des proto-catégories et la construction d'une collection de catégories disjointes. Chacune des deux est effectuée par un module de l'algorithme. Nous détaillons plus les deux étapes dans les paragraphes qui leur sont consacrés. Ici nous donnons seulement quelques indication nécessaires pour la compréhension globale de la démarche.

La première étape est la construction d'un ensemble de proto-catégories pour chacun des noyaux. Celle-ci se passe en deux sous-étapes: on constitue d'abord une liste de catégories *discriminantes* (cf. le paragraphe suivant) sur les SC membres. Une liste par couple de noyaux est formée. La suite consiste à faire des produits entre les catégories dans les différentes listes afin d'obtenir un ensemble de proto-catégories par noyau. Dans le pseudocode fourni à la fin de ce paragraphe, cette étape de l'algorithme est réalisée par le module CREER_Proto/SC. Sa description fait objet du paragraphe 2.3.1.

L'étape suivante consiste, dans une boucle interne, à choisir des collections de proto-catégories, une par noyau, et de les modifier. Le but des modifications est d'obtenir des catégories disjointes. La disjonction d'une collection de catégories est la tâche du module DISJONCTION_Proto/SC. Nous nous arrêterons avec plus des détails sur ce module dans le paragraphe 2.3.2.

Une fois la collection de catégories disjointes construite, elle est comparée à la meilleure collection trouvée jusqu'à lors. Suite à la comparaison, en termes du critère LEF, cette dernière est mise à jour. A la fin de la deuxième étape on dispose d'une partition qui sera associée à l'ensemble des noyaux courant. Dans la suite la condition d'arrêt sur le nombre des pas effectués est calculée. Au cas où celle-ci n'est pas vérifiée on procède au choix des nouveaux noyaux. Ce choix est fait en fonction de l'amélioration de la partition courante par rapport à celle au pas précédent. Les deux possibilités: individus centraux ou individus marginaux, ont été citées plus haut. Le pas itératif de la boucle principale se termine par le test d'une deuxième condition d'arrêt qui porte sur la stabilité de l'ensemble global des noyaux.

A la sortie de la boucle principale la meilleure partition rencontrée est fournie. Celle-ci est retournée en résultat par le module.

---

procedure **CLUSTER/SC-Partition** (

$E = \{id_i\}$	- ensemble d'individus
$SC = \{S_i\}$	- ensemble de systèmes classificatoires-éléments du produit
k	- le nombre des classes à construire.
b	- le nombre des pas de base
p	- le nombre des pas d'essai
r	- le nombre de nuées à essayer
$LEF = \{c_q, t_q\}$	- le critère d'évaluation lexicographique)

Structures de données:

Noyaux[k]	- un vecteur d'individus (les noyaux)
Nouveaux[k]	- un vecteur d'individus (les nouveaux noyaux)
Proto[k]	- un vecteur de listes de proto-catégories sur le produit des S_i
Espace	- une liste de k-uplets de catégories (l'espace des solutions)
Extension[k]	- un vecteur d'ensembles d'individus
Cour_disj[k]	- vecteur de catégories sur le produit
Cour_pas[k]	- vecteur de catégories sur le produit
Cour_glob[k]	- vecteur de catégories sur le produit

Variables scalaires:

bb = 0
pp = 0
amélioration = vraie
stop = faux
base = true

Algorithme:

 # Initialisation des noyaux pour les catégories à construire
Noyaux = *initialiser_noyaux*(k)
while not stop

```
Cour_glob = Ø

for i from 1 to k
  Proto [i] = CREER_Proto/SC(Noyaux, SC, i, k)
  Espace = Espace × Proto [i]                    //× dénote l'équivalent d'un produit Cartésien
endfor

# Choix de la meilleure combinaison de catégories disjointes
while choix(n, Espace)
  Cour_pas = Ø

  let n = (C1,C2,..Ck)
    for i from 1 to k
      Extension[i] = IA(Ci) ∩ E
    endfor
  endlet

  Cour_disj = DISJONCTION_Proto/SC(Extension, k)
  Cour_pas = evaluer({Cour_disj, Cour_pas}, LEF)

endwhile

# Comparaison par rapport à la qualité déjà ateinte
Cour_glob = evaluer({Cour_glob, Cour_pas}, LEF)
amélioration = (Cour_glob = Cour_pas)

# Partie contrôle global
if base then
  if amélioration then
    base = faux
    pp = 0
  endif
else
  pp = pp + 1
  if pp = p then
    bb = bb + 1
    if bb = b then
      stop = vrai
    endif
  endif
endif

# Choix des nouveaux noyaux
Nouveaux = calculer_noyaux(amélioration, Cour_pas)
if Nouveaux = Noyaux then
  stop = vrai
else
  Noyaux = Nouveaux
endif

endwhile

return(Cour_glob)
```

2.3.1. Module construction des proto-catégories

Le but du travail de ce module est la construction d'un ensemble de proto-catégories pour un noyau donné. Celle-ci doivent être discriminantes par rapport aux autres noyaux. La définition de la condition de *discrimination* suit.

Soit un système classificatoire $S=\langle C \leq \ll \rangle$ défini sur les langages L_C et L_I. Soient encore e et $e' \in L_I$ deux individus et $c \in L_C$ une catégorie.

Définition 24

La catégorie c est appelée discriminante pour e par rapport à e' si

$$e \in I_A(c) \text{ et } e' \notin I_A(c).$$

Les proto-catégories doivent donc vérifier la condition ci-dessus pour chacun des noyaux opposées. Afin de construire de telles catégories, l'algorithme se sert des projections sur les SC membres. Le principe est le suivant, on construit toutes les catégories discriminantes pour les projections des noyaux. Ensuite on effectue le produit entre un certain nombre de telles catégories de telle façon que le résultat de ce produit soit une catégorie qui discrimine le noyau par rapport à tous les autres noyaux.

Le traitement commence par une boucle sur toute paire de noyaux. Pour une telle paire on essaie de construire toutes les catégories discriminantes sur les SC membres. Pour cela on considère les projections des noyaux sur chacun de ces SC. Chacune des paires de projections est susceptible de donner lieu à une catégorie. Les seules exceptions en sont les projections identiques, c'est-à-dire les cas où les deux noyaux sont projetés vers le même individu. La construction d'une telle catégorie est effectuée par la primitive *écarter* dans le pseudocode ci-dessus. Une idée sur les catégories en question peut être trouvée sur Fig.28 (la vue est planaire). Elle montre les projections de trois noyaux avec la donnée de deux (sur six) catégories imaginables. Deux faits sont à noter. Premièrement, les catégories construites sont discriminantes pour un couple à la fois. Ainsi la catégorie *c1* discrimine l'individu *e1* par rapport à *e3* mais pas par rapport à *e2*. Deuxièmement, nous avons montré une situation irréaliste, les paires de projections étant considérées à part, sans égard aux autres projections de noyaux.

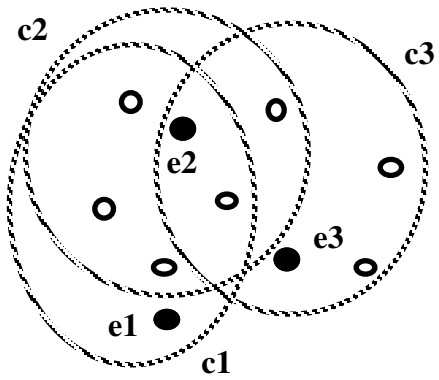


Fig.28. Exemple de catégories discriminant les noyaux sur un SC membre du produit.

A la suite de cette opération on dispose d'une liste de catégories par paire de noyaux. L'étape suivante consiste à combiner les catégories dans les différente listes qui discriminent un noyaux particulier. Une opération de distribution est définie (la primitive *distribuer*). Elle effectue le produit en prenant chaque fois une seule catégorie par liste. Ainsi on constitue un certain nombre de produits qui ont les propriétés suivantes:

- il sont constitués par exactement *k-1* catégories où *k* est le nombre des noyaux,
- comme catégories sur une projection partielle, mais non unitaire du produit global, ils sont discriminants sur le SC sous-jacent pour tous les noyaux.

Le pas suivant est de compléter chacun des produits ainsi construits de façon qu'il devienne une catégorie valide sur le produit global. Cette complétion consiste pour chacun des SC membres dont aucune catégorie n'est présente à rajouter au produit sa catégorie universelle. La complétion est confiée à la primitive *compléter* ci-dessus.

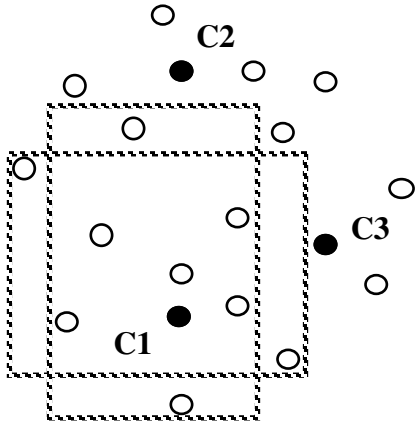


Fig.29. Exemple d'un ensemble de proto-catégories sur le produit pour la catégorie c1.

Le résultat global du travail est une liste de catégories du produit discriminantes pour un noyau par rapport au reste des noyaux, les proto-catégories. Une telle liste contient des proto-catégories dont l'interprétations peuvent varier. Tel est le cas des deux catégories visualisées sur Fig.29.

---

procedure CREER_Proto/SC (
E - ensemble d'individus (les noyaux)
SC - ensemble de systèmes classificatoires
i
k)

Structures de données:
El_Category [*k*] - le vecteur des "étoiles élémentaires", ensembles de sélecteurs

Variables scalaires:
ens_catégories - un ensemble d'ensembles de sélecteurs

Algorithme:

```
for j from 1 to k
  if i≠ j then
    for S in SC
      El_Category [j] = El_Category [j] ∪ écarter(E[i], E [j], S )
    endfor
  endif
endfor

partielles = distribuer(El_Category, i, k)
ens_categories = compléter(partielles, SC)

retourner(ens_categories)
```

2.3.4. Module disjonction de catégories

Le module travaille sur une collection de proto-catégories *a priori* non disjointes. Cela signifie que leurs interprétations forment un recouvrement sur l'ensemble des tous les individus. Par conséquent un certain nombre d'individus appartiennent à plusieurs catégories. La situation typique avant l'appel du module sur une collection de proto-catégories est illustrée par Fig.30. Le but est donc de modifier les catégories de façon à ce qu'elles deviennent disjointes. A noter que dans son travail l'algorithme prend en compte directement les ensembles d'individus associés aux catégories qui lui sont fournies en argument.

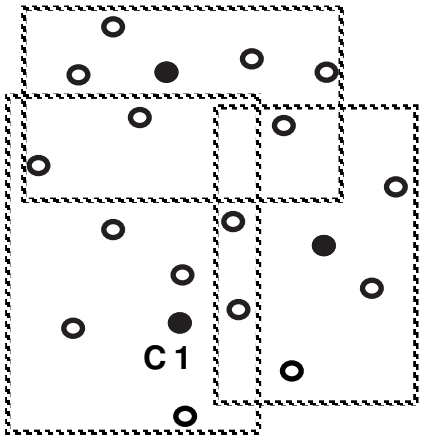


Fig.30. Une collection de proto-catégories non disjointes (vu planaire).

D'abord on procède par une première sélection des individus qui appartiennent à plusieurs catégories. Pour ce faire on parcourt les couples d'ensembles en calculant les intersections possibles et en accumulant les individus en question dans une liste (M-liste ci-dessus). A la fin de cette sélection chacun des ensembles ne contient que des individus qui lui sont propres. Ces individus sont utilisées pour former les nuées des futures catégories. L'opération sous-jacente est une généralisation (primitive Généraliser). Après cette opération nous nous retrouvons dans la situation illustrée sur Fig.31. Les nuées contiennent un sous-ensemble d'individus, le reste devrait être distribué sur la collection.

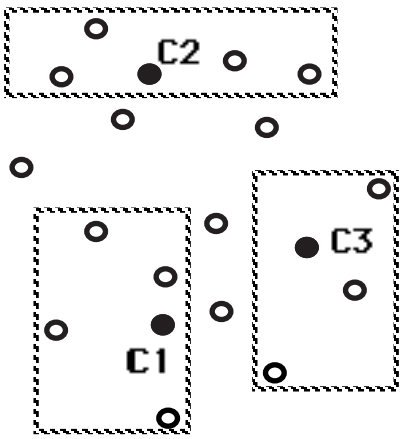


Fig.31. Les nuées de catégories de Fig.30.

Dans la suite on cherche à effectuer une distribution des individus de la liste de la meilleure façon possible. Pour des raisons d'efficacité l'optimum n'est pas visé. A la place, les individus sont traités un par un, et leur adjonction à chacune des nuées est considérée. Sur cet

procédure de base vue dans CLUSTER, nous avons appliqué une amélioration. Nous procédons à un tri de la liste des individus. L'ordre du tri est celui introduit par la moindre dissimilarité topologique par rapport à un des noyau. Une fois la liste triée, nous considérons ses éléments dans l'ordre. Pour chacun on compare les différentes collections que l'on obtient par l'adjonction de l'individu à une des nuées. En d'autres termes on forme les k collections possibles où chaque nuée à son tour intègre l'individu. Le reste des nuées ne changent pas. Les différentes combinaisons sont comparées moyennant le critère LEF et la meilleure est choisie. L'intégration de l'élément suivant continue à partir de cette combinaison. Par exemple, une étape intermédiaire de la distribution est montrée sur Fig.32. On peut s'apercevoir qu'il y a eu deux intégration réussites par rapport à la situation sur Fig.31.

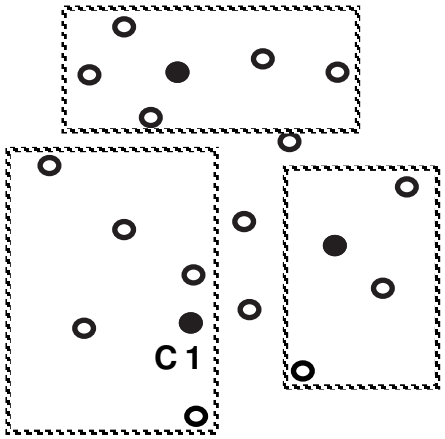


Fig.32. La distribution dans un stade avancé.

Une fois de plus, nous nous sommes donnés la liberté de modifier le procédé de base dans CLUSTER. Au lieu de considérer toutes les combinaisons possibles pour l'adjonction d'un individu, on se limite à un nombre préétabli (typiquement deux). Les catégories choisies pour cet ensemble réduit sont celles dont les noyaux sont les plus "proches" à l'individu. Cette modification a été justifiée par notre confiance en la mesure topologique. Elle est beaucoup plus adaptée à la construction des catégories par le fait qu'elle dépend des taxonomies dans les SC membres. Du coup on peut affirmer que plus la dissimilarité entre deux individus est élevée, plus leur généralisation est, dans le cas général, haut dans la taxonomie.

Une fois la distribution finie on se retrouve, avec un ensemble de catégories disjointes. Celles-ci ne sont plus des nuées car elles constituent une partition sur l'ensemble de tous les individus. Pour notre exemple, la situation finale est donnée sur Fig.33.

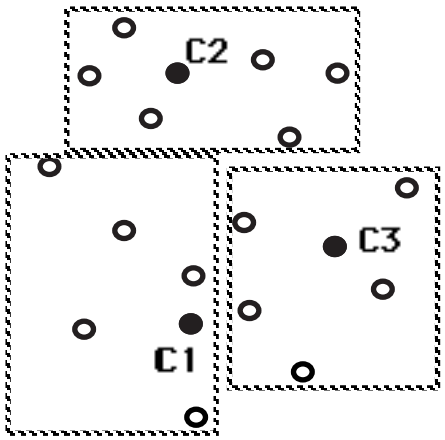


Fig.33. La collection après la fin de la distribution.

procedure DISJONCTION_Proto/SC (
Extension - vecteur d'ensembles d'individus
r
k)

Structures de données:
M-liste - la liste des individus appartenant à plusieurs interprétations
Catégorie[*k*] - vecteur de catégories
Catégorie_F[*k*]- vecteur de catégories (finales)
Nid[*k*, *k*] - table de catégories
Classe_F[*k*] - vecteur de catégories
Intersect - ensemble d'individus

Algorithme:

```
M-liste = Ø
for i from 1 to k
  for j from i+1 to k
    Intersect = Extension[i] ∩ Extension[j]
    M-liste = M-liste ∪ Intersect
  endfor
  Extension[i] = Extension[i] - M-liste
endfor

for i from 1 to k
  Catégorie[i]= Généraliser(Extension[i] )
endfor

trier(M-liste)

for e in M-liste
soient {e_i}_i=1,r les premiers r noyaux les plus proches à e
  for i from 1 to r
    soit l l'indice de e_i dans Noyaux
    for j from 1 to k
      if j = l then
        Nid[i,j]= Généraliser({Catégorie[j], {e})
      else
        Nid[i,j] = Catégorie[j]
      endif
    endfor
  endfor
  Catégorie_F = Evaluer({Nid[i]}, LEF)
endfor

return (Catégorie_F)
```

3. Conclusions

Dans ce chapitre nous avons définit un algorithme de construction de taxonomies. Il utilise comme formalisme de représentation le modèle des SC. Ceci lui permet de prendre en compte tous les types de données exprimés sous forme de systèmes classificatoires. Or nous avons montré comment les ADT et les concepts dans TROPES peuvent être modélisés par des

SC. Par conséquent, notre algorithme est capable d'inférer une taxonomie de classes sur n'importe quel concept TROPES. Pour ce faire, il est capable de descendre plusieurs niveaux dans le graphe des dépendances et de construire les taxonomies nécessaires. A chaque niveau, les taxonomies inférées seront utilisées lors du travail au niveau immédiatement supérieur. L'algorithme est donc complètement intégré dans le modèle TROPES et adapté à son système de typage ouvert.

Quels sont les avantages de notre algorithme du point de vu de la catégorisation conceptuelle. Premièrement, de par sa généralité, l'algorithme est parfaitement adapté au traitement des objets composites. En d'autre termes la structure dans la connaissance est prise en compte de manière très naturelle. Ceci constitue un fort contraste avec l'approche de CLUSTER/S [Stepp+ 86] par exemple qui tente aussi de rendre compte de cette structure. Tandis que ce dernier algorithme effectue un pas d'aplatissement et de cette façon "ramène" la totalité de l'information au niveau le plus haut, notre approche est de loin plus simple. Nous ne descendons dans la structure des objets qu'à un niveau et les calculs de dissimilarité et généralisation exploitent les taxonomies à ce niveau là. Par rapport à un autre algorithme qui traite la structure dans la connaissance, KBG [Bisson 92], notre algorithme présente l'inconvénient de ne pas travailler avec des relations circulaires. Mais l'algorithme en question ne possède pas la stratégie gagnante qui consiste à réduire la complexité du traitement en utilisant les taxonomies. Sans faire d'aplatissement comme dans CLUSTER/S, il travaille avec la totalité de l'information concernant un individu: dans la généralisation et dans le calcul de la dissimilarité.

Autre avantage de l'algorithme, c'est sa capacité de travailler avec des multiples points de vues sur un concept. A ce jour, ceci n'est qu'un témoignage de son intégration dans TROPES. Mais le maintient de multiples perspectives est un avantage que demain les modèles à objets seront obligés de fournir à leurs utilisateurs. Sur ce point nous avons pris aussi un légère avance car à notre connaissance aucune des méthodes existantes ne travaille dans un contexte multi point de vue.

L'algorithme CLUSTER/SC a été implémenté au sein du système TROPES (en ILOG TALK v.3.11. sur station de travail UNIX). L'implémentation utilise les primitives de dissimilarité, généralisation et couverture décrites dans le chapitre précédent. Un exemple d'application de notre algorithme sur une base de connaissances expérimentale est fourni dans l'Annexe D.

Conclusions et perspectives

L'inférence de taxonomies est une opération d'un grand intérêt en RCO car elle permet de mieux structurer une base de connaissances. Mais les méthodes de construction existantes sont dépassées par les derniers développements dans le domaine des objets. En particulier, ces méthodes ne sont pas adaptées aux systèmes extensibles qui permettent de caractériser les objets à l'aide de types données externes. De plus, ce genre de procédés ne prend pas en compte au degré approprié la taxonomie sur un domaine. Ainsi, elles passent à coté d'une expression très synthétique et, ce qui est plus important, universellement présente, de la structure du domaine. La conviction qu'un algorithme d'inférence de taxonomies doit obligatoirement s'appuyer sur les taxonomies existantes dans les sous-structures d'un domaine d'objets a été le point de départ de notre travail. Mais nous l'avons concrétisé dans un principe plus strict encore: un tel algorithme ne doit considérer d'autres informations sur un domaine que celles "codées" dans la taxonomie. Ceci est la clé pour la généricité d'un traitement sur des domaines de types variés car une taxonomie est toujours soit disponible, soit constructible.

Soucieux d'exploiter au maximum ces idées, nous avons fait appel à un modèle abstrait, les systèmes classificatoires, qui permet de représenter une taxonomie quelconque indépendamment du domaine sous-jacent. En se limitant sur les éléments essentiels tels l'ordre partiel entre catégories et l'appartenance d'un individu à une catégorie, les systèmes classificatoires restent génériques et leur application à un domaine concret ne demande qu'un minimum d'informations. Muni des opérations de produit et de projection, le modèle est capable de rendre compte de la structure des objets et, plus généralement, d'intégrer toute une base de connaissances à objets avec les relations entre ses domaines différents. En particulier, les SC offrent un cadre unificateur pour les concepts, ADT et contraintes dans TROPES. Leur universalité se transmet donc automatiquement sur tout traitement qui leur est intégré, ceci est également en vigueur pour l'inférence de taxonomies, même si le modèle n'a pas été conçu avec cette idée.

Dans leur définition d'origine, les SC considèrent les taxonomies comme des données *a priori*. Par conséquent, l'intégration de méthodes de construction automatique a nécessité que certaines extensions soient faites au modèle. Ainsi, nous avons défini la dissimilarité topologique, une mesure complètement intégrée dans le paysage des SC, jusqu'à lors très peu numérique. Sur un plan global la dissimilarité topologique est une mesure qui s'applique à tout domaine muni d'une taxonomie. Elle a l'avantage de n'utiliser qu'une structure vraiment minimale sur les domaines considérés. Et pourtant, la forme concrète sous laquelle cette mesure s'exprime dans les cas les plus courants (domaines totalement ordonnés, domaines énumérés) coïncide avec ce qui est globalement admis. Mais sa force se manifeste pleinement là ou d'autres approches ne sont pas convainquantes, à savoir sur les domaines structurés en un ordre partiel. Ceci est typiquement le cas des concepts ainsi que de certains types plus exotiques comme les ensembles et les listes.

La dissimilarité topologique avec le reste des primitives que nous avons introduit dans le modèle des SC ont rendu possible la conception de méthodes de construction directement en termes de SC. Ainsi, nous avons pu décrire un algorithme concret qui tient compte des principes énoncés ci-dessus. Notre algorithme hérite une partie de sa démarche, notamment lors de la construction d'un niveau de la taxonomie à CLUSTER qui a montré la plus forte cohésion avec les principes des SC parmi les méthodes examinés. Pour sa part, l'algorithme proposé est capable de travailler avec plusieurs SC qu'il traite de manière ascendante. Sont munis d'une taxonomies d'abord les SC modélisant des concepts primitifs, puis ceux qui les réfèrent et ainsi

de suite. Cette démarche permet une exploitation plus efficace des taxonomies sur les facteurs d'un SC produit, qu'elles soient construites par l'algorithme ou déjà disponibles. Ainsi, lors du traitement d'un niveau dans la structure des objets, la descente obligatoire dans les niveaux inférieurs n'a qu'une profondeur d'un. De par sa définition en termes de SC, l'algorithme possède un autre avantage, à savoir sa capacité de prendre un compte les domaines en n'utilisant qu'un minimum d'informations sur la nature de ceux-ci. Ainsi, en ne s'appuyant que sur les primitives intégrées au modèle, il profite au maximum des taxonomies disponibles tout en restant générique. Enfin, comparé aux autres méthodes de construction de taxonomies, notre algorithme se montre plus respectueux pour les ordres partiels qui existent sur les domaines.

Une version préliminaire de l'algorithme a été implémentée dans le cadre du système TROPES. Un objectif à plus long terme est d'intégrer l'algorithme au sein de T-TREE, la plate-forme d'inférence de taxonomies de TROPES [Euzenat 93b]. Nos projets sont d'enrichir celle-ci par d'autres méthodes de construction de taxonomies adaptées aux objets. La plate-forme inclut à ce jour un ensemble d'algorithmes numériques, dont une version de la méthode du plus proche voisin décrite précédemment. Nous envisageons donc l'intégration également de la dissimilarité topologique de façon qu'elle puisse être utilisée par les méthodes numériques. D'autres applications de la mesure, en dehors de la construction de taxonomies, peuvent être imaginées. D'une façon générale, elle peut trouver une place partout où une comparaison entre objets est effectuée. Par exemple, au sein d'un module d'apprentissage ou de raisonnement par cas. Il faut souligner que notre mesure ne devient pas une obligation dans le modèle des SC. Un utilisateur peut très bien la désactiver en faveur d'une dissimilarité *ad hoc* qu'il définit lui-même. Mais notre mesure constitue une possibilité universelle d'évaluer les ressemblances entre les individus, une sorte de dissimilarité "par défaut".

Sur le plan théorique, plusieurs axes peuvent être tracés. Ainsi, nous allons continuer avec l'étude des SC sur des types construits. Divisés en groupes comme nous l'avons fait dans le Chapitre III, ceux-ci ont des intérêts différents pour notre modèle. Les types construits existant dont le type de base est un ADT (comme les ensemble et les listes) vont nous guider dans la définition d'un constructeur abstrait de types (ADC). Les types construits sur des domaines d'objets nous éclaireront à propos de l'introduction de la notion de constructeur de SC. Celui-ci devrait s'exprimer à travers des nouveaux types de produit d'un ensemble de SC. Enfin, en intégrant des nouvelles notions au sein du modèle, nous serons, éventuellement, amenés à modifier la définition de la dissimilarité topologique. Une des modifications les plus probables sera d'y faire apparaître la cardinalité des catégories considérées.

Bibliographie

[Aguirr 89] Aguirre J.L. Construction automatique de taxonomies à partir d'exemples dans un modèle de connaissances par objets. Thèse de doctorat de l'INPG. Grenoble, 1989.

[Benkar 93] Benkaraache T. Problemes de validite en classification et quelques generalisations aux ultrametriques a valeurs dans un ensemble ordonne. These de Doctorat de l'UJF. Grenoble. 1993.

[Bisson 92] Bisson G. Conceptual Clustering in a First Order Logic Representation. Proceedings of 10th ECAI. pp. 458 - 462. Vienna, 1992

[Bisson 94] Bisson G. Conceptual Clustering. Machine Learning Summer School. Paris, 1994

[Bisson 95] Bisson G. Why and How to Define a Symilarity Meaure for Object-Based Representation Systems, in N.J.I.Mars (ed.) *Towards Very Large Knowledge Bases*, pp.236-246, IOS Press, Amsterdam,1995.

[Bouch+ 90] Bouchon-Meunier B., Desprès S., Dubois D., Gasquel O.,Guenoche A., Prade H. Interface entre Symbolique et Numérique. *Actes des 3° journées nationales PRC-GDR Intelligence Artifficielle*. pp. 89 - 109, Hermès, 1990.

[Bour+ 80] Bouroche J.M., Saporta C. L'Analyse de données. *Coll. "Que sais-je?"*, Paris, 1980.

[Buntine 88] Buntine W. Generalized Subsumption and its Application to Redundancy. *Artificial Intelligence* 36 (1) 1988, pp.149 - 176.

[Cappo+ 95] Capponi C., Euzenat J., Gensel J. Objects, Types and Constraints as Classification Schemes. *Procedings of the international symposium on Knowledge Retrieval, Use, and Storage for Efficiency*, Santa-Cruz (CA US), (11-13 Août) 1995 to appear.

[Capponi 95] Capponi C. Identification et exploitation des types dans un modèle de représentation des connaissances par objets, Thèse de doctorat, université Joseph-Fourier, Grenoble (FR), 1995 en préparation

[Carré+ 94] Carré B., Ducorneau R., Euzenat J., Napoli A., Rechenmann F. Classification et objets: programmation ou représentation? *Actes 5ièmes journées nationales du PRC-GDR «intelligence artificielle»*, Nancy (FR), pp213-237, (1-3 février) 1995

[Cheese+ 90] Cheeseman P., Kelly J., Self M., Stutz J., Taylor W., Freeman D. AutoClass : A Bayesian Classification System. *Reading in Machine Learning*. pp. 296 - 306. Morgan Kaufmann Publ., 1990.

[Clancey 84] Clancey W.J. Classification Problem Solving. *Proceedings of the National Conference on Artificial Intelligence*. Morgan Kaufmann, Austin, Texas, 1984, pp.49 - 55.

[Decaest 93] Decaestecker C. Aprentissage et outils statistiques en classification conceptuelle incrémentalle.*Revue d'Intelligence Artificielle* . Vol 7, num 1, pp. 33 - 71, 1993.

[Diday+ 76] Diday E., Simon J. C., Clustering Analysis. *Communiction and Cybernetics 10*, K.S. Fu (Ed.), Springer Verlag, Heidelberg, New York, 1976.

[Diday+ 82] Diday E., Lemaire J., Pouget J., Testu F. Eléments d'analyse de données. Dunod, Paris. 1982.

[Diday 93] Diday E. Quelques aspects de l'analyse des données symboliques. Rapport de recherche INRIA n° 1937, août 1993.

[Espos+ 91] Esposito F., Malerba D., Semeraro G. Flexible matching for noisy structural description *Actes 12th IJCAI*, Sydney (AU), pp658-664, 1989.

[Esposit 94] Esposito F. Conceptual Clustering in Structured Domains: a Theory Guided Approach in (E.Diday, Y.Lechevallier, M.Schader, P.Bertrand, B.Burtschy eds.) *New Approaches in Classification and Data Analysis*. pp. 395 - 404. Springer Verlag, Berlin, 1994.

[Euzen 93a] Euzenat J. Une définition abstraite de la classification et son application aux taxonomies d'objets, *Actes 2ndes journées «représentations par objets»* (RPO), La Grande Motte (FR), pp235-246, (17-18 juin) 1993

[Euzen 93b] Euzenat J. Brief overview of T-TREE : the TROPES Taxonomy building Tool. *Proc. 4th ASIS SIG/CR classification research workshop*, Columbus (OH US), pp. 69 - 87, 1993.

[Euzen 94] Euzenat J. Classification dans les représentations par objets: produits de systèmes classificatoires *Actes 9ième RFIA*, Paris (FR), pp185-196, (11-14 janvier) 1994

[Euzen 95] Euzenat J. Classification schemes: definitions, properties, algorithms internal report, INRIA Rhône-Alpes, Grenoble (FR), 1995, in preparation

[Fisher 87] Fisher D. Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning 2* pp. 139 - 172 . Kluwer Academic Publ., Boston, 1987.

[Forgy 65] Forgy E. Cluster analysis and multivariate data: efficiency versus interpretability of classifications.*Biometrics 21*, 1965

[Gabriel 94] Gabriel J.M. Définition de la notion de similarité dans les modèles à objets. Memoire de DEA de sciences cognitives, INP, Grenoble (FR), (30 juin) 1994.

[Gascuel 95] Gascuel O. Représentations arborées des proximités.Applications, propriétés et algorthmes. *Actes de JAVA'95* . pp. 41 - 49, Grenoble, 1995.

[Gensel 95] Gensel J. Integrating constraints in an object-based knowledge representation system. *Lecture notes in computer science* 923:67-77 (Manfred Meyer (éd.), Constraint processing), 1995.

[Gowda+ 94] Gowda K. C., Diday E. Symbolic Clustering Algorithms using Similarity and Dissimilarity Measures in (E.Diday, Y.Lechevallier, M.Schader, P.Bertrand, B.Burtschy eds.) *New Approaches in Classification and Data Analysis*. pp. 414 - 422. Springer Verlag, Berlin, 1994.

[Ichino+ 86] Ichino M. Pattern Classification Based on the Cartesian Join System: A General Rule for Feature Selection and Identification. *Proc. IEEE Int. Conf. Syst. Man Cybern.*, Atlanta, Georgia 1986, pp.14 - 17.

[Larson 77] Larson J. Inductive Inference in the Variable Valued Logic System VL21. Ph.D. Thesis. Dept. of Comp.Science, Urbana, 1977.

[Mariño+ 90] Mariño O., Rechenmann Fr., Uvietta P. Multiple perspectives and classificational mecanism in object-oriented representation. *Proc. 9th ECAI*, Stockholm pp. 425 - 430, 1990.

[Mariño 91] Mariño O. Classification d'objets composites dans un système de représentation de connaissances multi-points de vue. *Proc. 8th RFIA*, Villeurbanne, pp. 233 - 242 , 1991.

[Mariño 93] Mariño O. Raisonnement classificatoire dans une représentation à objets multi-points de vue. Thèse d'Informatique, Université Joseph-Fourier, Grenoble, octobre 1993.

[Mich 83] Michalski R.S. A Theory and Methodology of Inductive Learning. *Artifficial Intelligence 20* (1983), pp.111 - 116.

[Mich+ 84] Michalski R.S., Stepp R.E. Learning from Observation: Conceptual Clustering (Michalski, Carbonell and Mitchell, eds.) in *Machine Learning: An Artifficial Itelligence Approach I* ,pp. 331 - 363. Springer Verlag, Berlin, 1984.

[Napoli 92] Napoli A. Représentations à objets et raisonnement par classification en Intelligence Artifficielle. Thèse d'état de l'Université de Nancy I, 1992.

[Saporta 90] Saporta G., Probabilités, analyse de données et statistique. Editions Technip,1990.

[Sherpa 95] Sherpa project, TROPES 1.0 reference manual, internal report, INRIA Rhône-Alpes, Grenoble (FR), 1995

[Stepp+ 86] Stepp R.E., Michalski R.S. Conceptual Clustering: Inventing Goal-Oriented Classification of Structured Objects (Michalski, Carbonell and Mitchell, eds.) in *Machine Learning: An Artifficial Itelligence Approach II* pp. 471 - 498. Morgan Kaufmann Publ., Los Altos, 1986.

[vCutsem 94] van Cutsem B. (ed.). Classification and Dissimilarity Analysis. *Lecture Notes in Statistics*, Springer Verlag New York, Berlin, 1994.

[Valtchev 94] Valtchev P. Construction automatique de taxonomies à partir d'une collection d'objets. Mémoire de Magistère d'Informatique. , Université Joseph-Fourier, Grenoble, octobre 1994.

Annexe A

Le calcul des prédicats annotés.

Annotated Predicat Calculus (APC) [Mich 1983]
L'APC est un extension de la logique du 1° Ordre. C.à.d que tout ce qui est exprimable (et vrai) en LPO est aussi exprimable en ACP. En plus les notions suivantes ont été rajoutées:

Termes Composés (compound terms) par opposition aux termes élémentaires, ils peuvent comporter de :
. disjonctions internes - |
. conjonctions internes - &
qui ne peuvent pas être interprétés à l'extérieur des prédicats dont ils font partie.

N.B. *De tels termes n'ont pas de valeur de vérité*

Ex.

```
Bleu | Vert | Jaune ;  e1 & e2 & e3
```

Prédicats Composés (compound predicates) la seule différence avec les prédicats "classiques" est que au moins un des termes du prédicat est composé.

Ex.

```
Couleur( Boîte1, Bleu | Vert | Jaune )
Hauteur( e1 & e2 & e3, moyenne )
```

L'interprétation "classique" :

```
Couleur( Boîte1, Bleu | Vert | Jaune ) <=>
Couleur( Boîte1, Bleu ) | Couleur( Boîte1, Vert ) |
Couleur( Boîte1, Jaune )

Haut( e1 & e2 & e3, moyenne ) <=>
Haut( e1, moyenne ) & Haut( e2, moyenne ) &
Haut( e3, moyenne )
```

Expressions relationnelles (selectors)

Term1 rel Term2 où Term1, Term2 sont des termes (composés ou non)
et rel est dans { =, #, <, <=, >, >= }

- atomiques: Term1 rel a , où a est une constante d'un domaine quelconque

- générales: = non-atomiques

N.B. *Les prédicats peuvent être exprimés à l'aide de sélecteurs.*

Ex.

```
[ Nb_Pieds(Table1) = 4 | 6 ],
```

```
[ Diagnose = Normal | Non-conclusif ],

père(Laurent, Jacques) <=>
[ père(Laurent, Jacques) = true ]
```

La relation de "couverture"
Cette relation est équivalente à une implication et/ou réformulation entre 2 expressions en ACP: on dit que E1 "couvre" E2 si une expression E1' existe telle que E1' est équivalente à E1 à une réformulation près et E2 => E1' . La "couverture" est largement utilisée lors de la construction de l'extension d'une classe à partir de sa description.

Ex.

```
E1 =   [ Couleur = rouge | vert ] & [ Forme # grand ]
E2 =   [ Couleur = vert ] & [ Forme = petit ] &
       [ Nb parties = 3 ]
```

Dans ce cas la reformulation nécessaire est :

```
[ Forme # grand ]  <=>  [ Forme = petit | moyen ]
```

Annexe B

La dissimilarité sur les types construits.

L'Annexe B est divisé en deux grandes parties. Dans la première partie nous adoptons l'hypothèse que la modélisation d'un type construit est réalisée à l'aide d'un SC unitaire. Par opposition, nous considérons dans la partie deux que nous avons affaire à un sorte de produit de systèmes classificatoires dont la nature est encore à définir.

Pour la suite nous prenons les notations suivantes. D'abord nous supposons que les SC manipulés modélisent des types construits sur un type de base T dont le domaine est D. Nous allons également noter les individus des SC considérés par des lettres minuscules (a, b, etc.), tandis que leurs interprétations seront notées par des majuscules (A, B, etc.).

1. Ensembles et listes comme des SC ordinaires

Dans ce paragraphe nous considérons que les SC que nous manipulons ne sont pas des produits. Dans la suite nous considérons d'abord les types construits comme des ensembles. Nous commençons par une structure plutôt traditionnelle pour notre treillis où la généralisation de deux ensembles est leur intersection. En d'autres termes le langage des catégories est équivalent à 2^D, l'ensemble des sous-ensembles de D. Ensuite, nous montrons comment ce cas trop grossier peut être affiné par l'ajout de contraintes de cardinalité des individus dans l'interprétation d'une catégorie. Une troisième possibilité qui tient compte non pas des cardinalités mais des éléments dans les ensembles interprétations sera présentée. A la fin nous indiquerons seulement quel sera le résultat de la superposition des deux dernières variantes.

Les listes ne seront pas traitées de façon exhaustive. En ce qui concerne la fonction d'écart, leurs propriétés peuvent se rapporter, à peu de choses près, à celles des ensembles. Nous nous contenterons donc de n'en donner que les indications sur les différences.

1.1. Variante intersection

Pour une première variante nous prenons le cas le plus simple. La généralisation de deux ensembles sera leur intersection:

- Gen* $(a,b) = \|A \cap B\|$ pour deux individus a et b.

Le critère de sous-catégorisation $\ll$ sera pour nous l'inverse de l'inclusion ensembliste et le langage des catégories L_C sera l'ensemble 2^D. Nous avons donc un isomorphisme entre les deux structures d'ordre partiel qui sont en effet de treillis.

- $\langle L_C \ll \rangle \cong \langle 2^D \supseteq \rangle$.

Tout d'abord, nous allons étudier une expression de la fonction d'écart (non normalisée) δ'et de la fonction δ'_n (normalisée).

Lemme 7

δ' peut être calculée comme $|A \Delta B|$ d'où $\delta'_n(A,B) = \frac{|A \Delta B|}{\delta'_{max}}$ (la différence symétrique normalisée).

Ci-dessous nous donnons l'exemple d'un domaine *D*.

Ex. Soit $D = \{a, b, c, d, e, f, g, h\}$
Gen* $(\{a, d, f, g\},\{a, b, d, e\}) = \|a, d\|$
$\delta'_n(\{a, d, f, g\},\{a, b, d, e\}) = \frac{4}{8} = \frac{1}{2}$
Gen* $(\{c, f\},\{b\}) = \| \|$
$\delta'_n(\{c, f\},\{b\}) = \frac{3}{8}$
Gen* $(\{a, e, f\},\{a, f\}) = \|a, f\|$
$\delta'_n(\{a, e, f\},\{a, f\}) = \frac{1}{8}$

Nous pouvons maintenant calculer la valeur à utiliser pour la pondération.

Lemme 8

δ'_{max} est la cardinalité du domaine, $|D|$.

Ici un problème peut apparaître en cas de domaine infini. Quand le domaine *D* est *a priori* infini, c'est-à-dire quand $|D| = +\infty$ nos calculs ne peuvent plus se servir de la cardinalité. Pour y remédier nous effectuons une réduction sur les valeurs effectivement présentes dans la base. En résultat on obtient *D'*, un sous-ensemble fini de *D*. Cette réduction nous permet de n'opérer qu'avec des quantités finies. Il est à noter que cette substitution de la cardinalité réelle par une autre valeur ne change pas l'ordre respectif des dissimilarités à l'intérieur de *D'*.

Théorème 9

δ'_n est une distance.

Quels sont les problèmes que nous voyons dans cette première variante sur les ensembles qui est basée exclusivement sur l'intersection. Premièrement, une petite confusion est à éviter, les catégories est les individus sont deux langages identiques par leur expression. Pour cette raison nous avons employé les notations $\| \|$ pour une catégorie. Ceci est un inconvénient secondaire. Un problème plus important est la qualité de la généralisation dans cette première variante. Nous l'avons trouvé trop grossière. Voici un exemple qui montre l'envergure de l'interprétation abstraite d'une catégorie.

Ex. $I_A(\|5\|) = \{\{5\},\{5, 1\},\{5, 2\},.....\} = 2^{D - \{5\}}$

Nous en concluons que la généralisation se situe trop haut dans la taxonomie. Du coup, elle a une interprétation qui comporte trop d'individus. Ensuite, la mesure δ' donne une vision trop égalitaire sur les différences entre ensembles de valeurs. Elle ne prend pas en considération le sous-ensemble des valeurs partagées. En d'autres termes, l'union des deux ensembles n'intervient pas dans les calculs.

Ex. Avec le même domaine que plus haut, $D = \{a, b, c, d, e, f, g, h\}$

Soient les trois ensembles A,B et C comme suit: $A = \{a, d, f, g\}$, $B = \{a, b, d, e\}$, $C = \{a, c, d, e, f, h\}$.

Les deux écarts sont égaux $\delta'_n(A,B) = \frac{4}{8}$ et $\delta'_n(A,C) = \frac{4}{8}$ aussi. Pourtant, intuitivement, on aura tendance à considérer la proximité entre A et C supérieure à celle entre A et B. Un argument en faveur de la mesure consiste à insister sur les disparités de cardinalité entre les ensembles, qui dans le cas du couple (A,C) doivent compenser en quelque sorte le partage d'un nombre plus grand de valeurs.

1.2. Intersection plus contraintes de longueur

Comme précédemment nous définissons le langage des catégories L_C qui cette fois-ci inclut des contraintes de cardinalité sur les ensembles membres d'une interprétation. En d'autres termes

- $L_C \subseteq 2^D \times \Im$, ($\Im$ est la famille des intervalles inclus dans $[1..|D|]$).

A noter l'inclusion à la place de l'égalité. Elle se justifie par l'existence d'un certain nombre de combinaisons *absurdes*, c'est-à-dire des cas où une contradiction entre la cardinalité minimale de l'ensemble dans 2^D et celle donnée par l'intervalle dans $\Im$ existe. Ceci est discuté plus loin (cf. les exemples).

Ensuite, notre critère de sous-catégorisation devient le produit des ordres respectifs.

- $\ll = \supseteq_D \times \subseteq_\Im$, ($\subseteq_\Im$ équivaut l'inclusion ensembliste entre intervalles).

De même nous définissons la généralisation comme le produit entre l'intersection des ensembles et l'enveloppe des intervalles (cf. Définition 3 § 4.2.).

- Gen* $(a,b) = \|A \cap B\| \times [\min(|A|, |B|), \max(|A|, |B|)]$ pour deux individus a et b.

Enfin, la fonction d'écart est notée:

- δ'' est la *fonction d'écart topologique* sur le graphe du treillis $\langle L_C, \ll \rangle$.

Nous allons prouver que l'égalité suivante est vérifiée $\delta'' = \delta' + \delta_\Im$ où $\delta_\Im$ est la fonction d'écart définie sur $\langle \Im, \subseteq_\Im \rangle$ (cf. § 4.2.).

Théorème 12

$\delta'' = \delta' + \delta_\Im$. Chacune des deux composantes est calculée sur la partie correspondante de l'expression en L_C.

Ex. ($D \subseteq \mathrm{N}$, $D = \{1,..,10\}$)
Gen* $(\{1, 5, 6\},\{3, 5, 8, 9\}) = \|5\|, [3,4]$
$\delta''(\{1, 5, 6\},\{3, 5, 8, 9\}) = \delta''(\|1, 5, 6\|[3,3], \|3, 5, 8, 16\|[4,4]) = 5 + 2 = 7$
Gen* $(\{2, 5\},\{3\}) = \| \|, [1,2]$

$\delta"(\{2, 4, 5\},\{3\}) = \delta"(\|2, 4, 5\|[3,3], \|3\|[1,1]) = 4 + 4 = 8$
Gen* $(\{5, 7, 8\},\{7, 8\}) = \|7, 8\|, [2..3]$

Afin d'éviter les calculs supplémentaires nous proposons de calculer $\delta"_n$ directement à partir des fonctions normalisées: $\delta"_n = \frac{\delta'_n + \delta_{\Im n}}{2}$.

Ex.

$$\delta"_n(\{1, 5, 6\},\{3, 5, 8, 9\}) = \frac{\delta'_n(\{1, 5, 6\},\{3, 5, 8, 9\}) + \delta_{\Im n}([3,3],[4,4])}{2} =$$

$$= \frac{0,5 + 0,1}{2} = 0,3$$

$$\delta"_n(\{2, 4, 5\},\{3\}) = \frac{0,4 + 0,2}{2} = 0,3$$

Théorème 13

$\delta"$ et $\delta"_n$ sont des distances.

Il existe un nombre de combinaisons redondantes dans $2^D \times \Im$. Ce sont des typiquement des éléments où les deux composantes sont en désaccord. Nous en donnons quelques exemples avant d'énoncer un résultat sur le nombre total de tels éléments. Ensuite, un théorème démontrant que l'existence de tels éléments n'est pas gênante par rapport à nos calculs, est proposé.

Ex.
Un premier cas est illustré par l'expression $\|1, 5, 6\|[2,4]$. Ici il y a une contradiction entre la borne inférieure de l'intervalle sur la cardinalité, deux, et la cardinalité minimale effective, trois. En autres mots, aucun ensemble de cardinalité deux ne peut comporter en même temps *1, 5* et *6*. Dans ce cas l'interprétation abstraite de l'élément double celle d'un autre élément, en occurrence $\|1, 5, 6\|[3,4]$. Nous allons noter l'ensemble de ces combinaisons, que nous allons désormais appeler *redondants* avec Θ.
$\|3, 5, 8, 16\|[2,3]$ représente un deuxième type de problème, contenu dans le premier mais plus frappant encore. Maintenant, c'est la borne supérieure maintenant qui reste au-dessous de la cardinalité minimale effective. Il s'agit, en effet, d'un élément dont l'interprétation abstraite est vide. Pour de raisons de commodité, ces combinaisons seront notées comme *absurdes*. Leur ensemble sera Ξ. on pourra remarquer que $\Xi \subseteq \Theta$.

Avant de procéder avec la suite, introduisons également l'ensemble des combinaisons valables ou *admissibles*, Ω ($\Omega = 2^D \times \Im - \Theta$).

Il est maintenant légitime de s'interroger sur le rôle de ces éléments redondants. Car jusqu'à lors notre raisonnement ne les a pas pris en compte. Il est intéressant de savoir qu'on est capable d'évaluer leur nombre en fonction de la cardinalité de *D*. Avant de montrer ce résultat, notons que le nombre total d'éléments dans $2^D \times \Im$ est le produit de deux cardinalités 2^D et $|\Im|$. En admettant que $|D| = n$, la a cardinalité de $\Im$ s'élève à $\frac{n(n+1)}{2}$.

Assertion 14

Le nombre des combinaisons *redondantes* est exactement

$$\sum_{i=1}^{n} (n - i + 1) \sum_{j=i+1}^{n} \binom{n}{j} .$$

De la même façon on pourra calculer le nombre exact des éléments absurdes, c'est-à-dire ceux dont l'interprétation abstraite est vide. A titre d'information, il y en a

$$\sum_{i=1}^{n} i \sum_{j=1}^{i-1} \binom{n}{j} .$$

Il ne nous reste qu'à démontrer la validité des résultats énoncés précédemment dans le cas où les combinaisons redondantes sont écartées. Quel genre de problèmes une telle réduction peut-elle entraîner? On pourra s'attendre, par exemple, à ce que l'absence d'un certain nombre de nœuds dans le produit perturbe le calculs des plus courts chemins. Ou bien que, auparavant, la généralisation la plus spécifique de deux éléments admissibles (par opposition aux éléments redondants) soit redondante. Ceci n'est, heureusement, pas le cas comme le montrent les deux lemmes suivants.

Lemme 15

L'ensemble des éléments *admissibles* est fermé par généralisation. Autrement exprimé
$\forall A, B \in 2^D \times \Im$, si A et $B \in \Omega$, alors Gen*$(A,B) \in \Omega$.

Après avoir situé la généralisation de deux éléments, nous allons montrer qu'il existe toujours au moins un chemin de longueur minimale entre cette généralisation et chacun des deux éléments dont la totalité des sommets appartiennent à Ω. Par analogie avec les éléments, nous allons appeler ce type de chemins *admissibles*.

Lemme 16

Entre deux sommets comparables *admissibles*, il y a toujours un chemin à longueur minimale (calculée selon la formule (5.1) plus haut) *admissible*.

1.3. Variante intersection/union

Cette variante pend pour le langage des catégories L_C le double produit de 2^D. La première dimension dénote, comme avant, l'intersection, tandis que la seconde donne l'union des deux ensembles. L'idée de la deuxième dimension est de définir un ensemble de valeurs dont tous le individus de l'interprétation doivent être inclus.

- $L_C = 2^D \times 2^D$.

Le critère de sous-catégorisation est le produit des deux inclusions ensemblistes, une inversée et une ordinaire.

- $<< = \supseteq_D \times \subseteq_D$

La généralisation de deux individus est le produit entre l'intersection des premiers éléments du produit et l'union des seconds.

- Gen* $(a,b) = \|A \cap B\| \times \{A \cup B\}$ pour deux individus a et b.

Dans ces condition la fonction d'écart est notée:

- δ''' est la *fonction d'écart topologique* sur le graphe du treillis $\langle L_C, \ll \rangle$.

Nous allons prouver l'égalité suivante $\delta''' = \delta' + \delta_\tau$ où δ_τ est la *dissimilarité topologique* sur $\langle 2^D, \subseteq_D \rangle$. δ_τ possède de propriétés analogues aux propriétés de δ' sur $\langle 2^D, \supseteq_D \rangle$. Ceci est immédiat, car la valeur est dans les deux cas la même: la cardinalité de la *différence symétrique*. Ceci va sans doute faciliter son étude ainsi que l'étude de δ'''.

Après avoir remarqué la similitude entre les deux mesures, δ' et δ_τ, nous allons pouvoir donner la preuve de la formule exacte du calcul de δ'''.

Lemme 17

δ''' peut être calculée en somme de δ' et δ_τ, c'est-à-dire que

$$\delta''' = \delta' + \delta_\tau.$$

Ex. ($D \subseteq$ N, $D = \{1,..,10\}$)
Gen* $(\{1, 5, 6\},\{3, 5, 8, 9\}) = \|5\|, \{\{1, 3, 5, 6, 8, 9\}\}$
$\delta'''(\{1, 5, 6\},\{3, 5, 8, 9\}) = \delta''(\|1, 5, 6\|\{\{1, 5, 6\}\}, \|3, 5, 8, 16\|\{\{3, 5, 8, 9\}\}) = 5 + 5 = 10$
Gen* $(\{2, 5\},\{3\}) = \| \|, \{\{2, 3, 5\}\}$
$\delta'''(\{2,, 5\},\{3\}) = \delta'''(\|2, 5\|\{\{2, 5\}\}, \|3\|\{\{3\}\}) = 3 + 3 = 6$

Afin de pouvoir calculer δ'''_n nous avons besoin de la valeur de δ'''_{max}. Elle est fixée par le lemme suivant.

Lemme 18

δ'''_{max} est le double de la cardinalité du domaine, $2*|D|$.

Ex.

$$\delta'''_n(\{1, 5, 6\},\{3, 5, 8, 9\}) = \frac{\delta'''(\|1, 5, 6\|\{\{1, 5, 6\}\}, \|3, 5, 8, 16\|\{\{3, 5, 8, 9\}\})}{\delta'''_{max}} =$$

$$= \frac{5 + 5}{20} = 0,5$$

$$\delta'''_n(\{2,, 5\},\{3\}) = \frac{3 + 3}{20} = 0,3$$

A noter que δ'''_n et δ'_n sont identiques. Ceci est dû à la normalisation.

Théorème 19

δ''' et δ'''_n sont des distances.

Jusqu'à lors nous avons ignoré l'existence d'un certain nombre d'éléments *absurdes* dans $2^D \times 2^D$. Il est tout de même légitime de vouloir les extraire du langage L_C, car leur rôle n'est qu'à rendre un traitement plus complexe et donc plus coûteux. Le nouveau L_C, résultat de la réduction ainsi définie, risque d'invalider rendre certains des résultats énoncés précédemment. Dans la suite, notre but principal sera, comme il a été fait dans le paragraphe précédent, de montrer qu'une telle simplification n'est pas essentiels pour nos calculs. Tout d'abord nous donnons une définition de ce qu'est un élément *absurde*, ensuite nous montrons des résultats sur le nombre total de tels éléments.

Premièrement, il est clair que le nombre total d'éléments dans $2^D \times 2^D$ est $2^{2^{*|D|}}$. Parmi eux les éléments à une interprétation abstraite vide sont ceux dont la partie union ne comporte pas en entier la partie intersection. Plus formellement

Définition 25

Soit $A = E \times E'$ un élément de $2^D \times 2^D$. On dira que A est *admissible* si $E \subseteq E'$. Dans le reste des cas A sera noté comme *absurde*.

Comme précédemment, l'ensemble des éléments *absurdes* sera noté par Ξ et son complémentaire par Ω. A noter l'absence d'éléments *redondants*, contrairement au cas précédent.

L'étape suivante est consacré à l'évaluation de la cardinalité de Ξ. Nous proposons le résultat suivant

Assertion 20

Soit $|D| = n$. Le nombre des combinaisons *absurdes* est exactement

$$\sum_{k=0}^{n} \binom{n}{k} * (2^{|D|} - 2^{k}).$$

Le résultat suivant est l'équivalent du Lemme 15 du paragraphe précédant. Elle nous assure qu'on ne perde pas la propriété d'admissibilité par généralisation.

Lemme 21

Ω est fermé par généralisation. En outre,

$$\forall A, B \in 2^D \times 2^D, \text{ si } A \text{ et } B \in \Omega, \text{ alors } \mathrm{Gen}^*(A,B) \in \Omega.$$

Il reste à montrer maintenant que la dissimilarité δ''' est toujours valide dans le nouveau contexte. Ici encore, nous insistons sur le parallèle entre les résultats rapportés dans le paragraphe précédent et ceux que nous sommes en train d'établir. Sur ce point, le lemme ci-dessous est à comparer avec le Lemme 9. Il nous donne la certitude que parmi les chemins *admissibles* entre deux nœuds, il y a toujours des chemins de longueur minimale dans $2^D \times 2^D$.

Lemme 22

Entre deux sommets comparables *admissibles*, il y a toujours un chemin de longueur minimale *admissible*.

1.4. Variante intersection/union plus contraintes de longueur

Nous ne ferons que de signaler que la possibilité de combiner les deux dernières variantes existe. Les résultats qui peuvent être obtenus sont essentiellement les mêmes que dans le cas de contraintes de longueur.

1.4. Listes

Les listes sont différentes des ensembles sur deux points essentiels. Premièrement, ce sont des collections avec répétition. Il y a donc aucune limite *a priori* pour leur longueur, même si le domaine de base est fini. Mais dans tout les cas réels des listes d'une longueur finie et limitée sont considérées. Ensuite, il y a un ordre secondaire total sur les éléments d'un liste introduit par leur place dans la liste. Ces deux faits sont très importants quant à la définition de la généralisation sur listes qui, rappelons-le, est déterminante pour la structure de la taxonomie. Nous avons tendance, par rapprochement avec les ensembles, à considérer sur les listes une opération de la sorte de l'intersection. Cette dernière peut être vue comme la prise d'un sous-ensemble commun maximal. Malheureusement, dans le cas de deux listes, la sous-liste de longueur maximale n'est pas unique. Par conséquent la structure définie par la relation de sous-liste n'est pas un treillis.

Ce fait ne gêne que très peu la fonction d'écart topologique qui, comme nous l'avons indiqué lors de la définition, peut être généralisée dans un tel cas. Nous affirmons que la fonction d'écart sur deux listes peut être calculée comme une différence de longueurs de listes. Pour cela nous introduisons un opérateurs de généralisation qui fait apparaître l'ordre dans les listes. Celui-ci n'as pas beaucoup de sens en dehors de la définition de la fonction d'écart sur les listes.

Définition 26

La généralisation des deux listes l_1 et l_2 est :

$$\text{Gen}^*(\, l_1, l_2\,) = l$$

Où l est la sous-liste de l_1 et l_2 de longueur maximale située le plus à gauche dans l_1.

Il convient de remarquer que la définition n'est pas symétrique. En effet, le choix d'une généralisation particulière parmi toutes celles qui sont possibles est une question de commodité. Il simplifie les notation sans pour autant influencer le résultat final. Ce dernier est exprimé sous une forme numérique et est invariant sur tout facteur concernant la structure des listes.

Nous définissons maintenant la fonction d'écart δ^l comme suit

Définition 27

La fonction δ^l sur les deux listes l_1 et l_2 est :

$$\delta^l(\, l_1, l_2\,) = \text{length}(l_1) + \text{length}(l_2) - 2*\text{length}(l)$$

Où l est la sous-liste de l_1 et l_2 telle que $l = \text{Gen}^*(\, l_1, l_2\,)$.

La fonction définie ci-dessus coïncide avec la fonction d'écart sur les listes dans le cas où la généralisation est celle donnée par la définition 26. Ce fait ne va pas être démontré pour

des raisons de simplicité. Notons tout de même qu'il admet une preuve semblable à celle du Lemme 7. De même la valeur maximale de cette fonction est le double de la longueur maximale d'une liste (à condition qu'une telle est fixée). Pour s'apercevoir, il suffit de prendre deux listes de longueur maximale, chacune composée par les répétitions d'un élément qui varie dans les deux listes.

Ex.
Soit le domaine des lettres $D = \{a, b, c, d, e, f,\}$
Soient A,B et C sont les listes suivantes: $A = (a, b, a, c, b, d, a)$, $B = (a, c, b, a, d)$. Les deux généralisations sont: $Gen^*(A,B) = (a, b, a, d)$ et $Gen^*(B,A) = (a, c, b, a)$
Les dissimilarités sont: $\delta^l(A,B) = \delta^l(B,A) = 4$.

2. Produits de SC

D'origine nous avons voulu mettre ici des définitions de nouvelles sortes de produits des SC. Ceux-ci devraient correspondre aux constructeurs de types et permettre le comparaison entre individus. Ainsi la fonction d'écart devrait pouvoir s'appliquer avec succès à des cas d'ensembles d'objets dont la comparaison relève d'un appariement. Cet objectif s'est révélé de loin trop ambitieux vu le temps dont nous avons disposé pour finir et rapporter notre travail. Mais un continuation est envisagée. Elle pourra déboucher vers un enrichissement essentiel du modèle des SC. Pour l'instant nous pouvons fournir un exemple qui met en évidence la nécessité des nouveaux types de produit.

Ex.
Soit le domaine des couleurs $D = \{rouge, blanc, vert, bleu, lilas, jaune, orange, bordeaux\}$
Un désaccord entre l'intuition et le calcul peut être observé. Par exemple, si A,B et C sont les ensembles suivants: $A = \{rouge, blanc, orange\}$, $B = \{blanc, jaune, bordeaux\}$ et $C = \{blanc, vert, bleu\}$ et que nous nous servons de la fonction δ'_n définie dans § 6.2.1. alors les deux dissimilarités sont $\delta'_n(A,B) = \frac{2}{8} = \frac{1}{4}$ et $\delta'_n(A,C) = \frac{2}{8} = \frac{1}{4}$ respectivement. Malgré cette égalité, et par la proximité très naturelle entre les couples (*rouge*, *bordeaux*) et (*jaune*, *orange*) nous préférons donner une importance plus grande aux ressemblances entre A et B. Ce phénomène ne se présente pas si l'on cherche à traiter les collections de valeurs comme des produits de SC plutôt que de les considérer au même titre que les autres SC.

Annexe C

Preuve des propriétés du Chapitre III et annexe B.

Théorème 1

δ_n est un indice de dissimilarité.

Preuve:

Nous devrons démontrer que δ_n vérifie les trois propriétés de la définition d'une dissimilarité.

Pour la première, valeurs non négatives, il suffit de se rendre compte que les cardinalités des chemins sont des telles valeurs et que l'addition et la division préservent cette propriété.

La symétrie suit de la façon dont on calcule la fonction d'écart: l'addition est commutative.

La troisième propriété est aussi immédiate, les chemins entre une catégorie et elle-même ont une cardinalité zéro.

Par conséquent nous avons défini un indice de dissimilarité.

❑

Corollaire 1.1

La dissimilarité topologique est aussi un indice de dissimilarité.

Preuve:

Les deux premières propriétés sont immédiates, car l'addition sur les fonctions d'écart les préserve. Pour la troisième il faudra remarquer que deux individus sur un SC produit sont identiques si et seulement si toutes leur projections le sont.

❑

Fonction d'écart sur les intervalles.

Propriété 4

Pour deux intervalles $I = [a_1,b_1]$ et $J = [a_2,b_2]$ il y a au plus deux intervalles non vides dans $I \Delta_{\Im} J$.

Preuve:

Ceci peut être prouvé en considérant les différents cas de base à part, et notamment les cas d'intervalles disjoints, d'intersection non vide et d'inclusion complète. Nous proposons une approche générique qui consiste à considérer les résultats de $(I \otimes J)$ et $(J \otimes I)$. Chacune de ces

deux différences pris à part, elle peut comporter au plus deux intervalles, éventuellement vides. Sans aucune hypothèse sur ces intervalles considérons leurs écritures. Celles-ci sont:

$[\min(a_1,a_2),a_1\text{-}1]$ et $[b_1+1,\max(b_1,b_2)]$ pour $I \otimes J$,

$[\min(a_1,a_2),a_2\text{-}1]$ et $[b_2+1,\max(b_1,b_2)]$ pour $J \otimes I$.

Il est clair maintenant que au moins un des intervalles $[\min(a_1,a_2),a_1\text{-}1]$ et $[\min(a_1,a_2),a_2\text{-}1]$ est un intervalle vide, car $\min(a_1,a_2) < \min(a_1\text{-}1,a_2\text{-}1)$. Idem pour les deux autres intervalles. Ceci nous fait un total d'au plus deux intervalles non vides dans $I \Delta_{\Im} J$.

❑

Lemme 5

Soient $I = [a_1,b_1]$ et $J = [a_2,b_2]$ deux intervalles dans $\Im$. Considérons la longueur du plus court chemin dans le graphe du treillis $\langle \Im, \subseteq_{\Im} \rangle$ allant du sommet correspondant à I jusqu'à celui correspondant à $I \Delta_{\Im} J$. Cette longueur est égale à la cardinalité cumulée de $I \otimes J$.

Preuve:

De façon analogue, on compte les valeurs intermédiaires.

❑

Corollaire 5.1

Puisque $\text{card}(I \Delta_{\Im} J) = \text{card}(I \otimes J) + \text{card}(J \otimes I)$, et la longueur du chemin est additive, alors $\delta_{\Im}(I,J) = \text{card}(I \Delta_{\Im} J)$.

Ceci nous permet d'affirmer que, pour les intervalles $I = [a_1,\mathbf{b_1}]$ et $J = [a_2,\mathbf{b_2}]$, on a $\delta_{\Im}(I,J) = \text{card}([\min(a_1,a_2),a_1\text{-}1], [b_1+1,\max(b_1,b_2)], [\min(a_1,a_2),a_2\text{-}1], [b_2+1,\max(b_1,b_2)])$. En sachant que $\text{card}([a,b]) = b - a + 1$, nous pouvons conclure que:

$$\delta_{\Im}(I,J) = a_1 - \min(a_1,a_2) + a_2 - \min(a_1,a_2) + \max(b_1,b_2) - b_1 + \max(b_1,b_2) - b_2.$$

Ou encore:

$$\delta_{\Im}(I,J) = a_1 + a_2 - 2*\min(a_1,a_2) + 2*\max(b_1,b_2) - b_1 - b_2. \qquad (*)$$

❑

Théorème 6

$\delta_{\Im}$ est une distance.

Preuve:

Soient donnés les trois intervalles suivants $I = [a_1,b_1]$, $J = [a_2,b_2]$ et $K = [a_3,b_3]$. Sans perte de généralité nous pouvons considérer que $\delta_{\Im}(I,J) \geq \delta_{\Im}(I,K) \geq \delta_{\Im}(K,J)$. Il faut donc montrer que $\delta_{\Im}(I,J) \leq \delta_{\Im}(I,K) + \delta_{\Im}(K,J)$.

Pour cela nous divisons la formule globale en deux parties que nous allons traiter à part. Tout d'abord nous allons montrer l'inégalité:

$$2*\max(b_1,b_2) - b_1 - b_2 \leq 2*\max(b_1,b_3) - b_1 - b_3 + 2*\max(b_2,b_3) - b_2 - b_3 \quad (4.0)$$

En effet, après les simplifications (changement du côté, puis division par 2) nous obtenons:

$$\max(b_1,b_2) + b_3 \leq \max(b_1,b_3) + \max(b_2,b_3). \qquad (6.1)$$

Maintenant il faut remarquer le fait suivant. Quels que soient b_1,b_2 et b_3, sur les trois nombres: max(b_1,b_2), max(b_1,b_3) et max(b_2,b_3), les deux plus grands sont égaux. Ceci nous permet d'affirmer qu'au moins un des deux inégalités ci-dessous est valide:

$\max(b_1,b_2) \leq \max(b_1,b_3)$, (6.2)

$\max(b_1,b_2) \leq \max(b_2,b_3)$. (6.3)

Supposons que c'est (4.2) qui tient. Clairement:

$b_3 \leq \max(b_2,b_3)$. (6.4)

En ajoutant les deux inégalités (4.2) et (4.4) nous obtenons exactement (4.1). Le cas où (4.3) est à traiter de façon analogue. La validité de (4.1) entraîne celle de (4.0).

La deuxième partie de cette preuve consiste à démontrer l'inégalité complémentaire à (4.0), à savoir:

$a_1 + a_2 - 2*\min(a_1,a_2) \leq a_1 + a_3 - 2*\min(a_1,a_3) + a_3 + a_2 - 2*\min(a_3,a_2)$.(6.0')

Nous nous contenterons de remarquer que le raisonnement reste essentiellement le même. Seuls les signes changent, ainsi que le maximum est remplacée par le minimum. Il faut aussi noter que la propriété des trois doubles maxima se traduit par une égalité entre les deux minima les plus petits.

Finalement, en ajoutant les deux inégalités (4.0) et (4.0'), on obtient le résultat à prouver, et notamment, $\delta_{\Im}(I,J) \leq \delta_{\Im}(I,K) + \delta_{\Im}(K,J)$.

❑

Variante intersection

Lemme 7

δ' peut être calculée comme $|A \Delta B|$ d'où $\delta'_n(A,B) = \dfrac{|A \Delta B|}{\delta'_{max}}$ (la différence symétrique normalisée).

Preuve:

Il faut montrer que la *dissimilarité topologique* pour deux ensembles est proportionnelle à la cardinalité de leur différence symétrique.

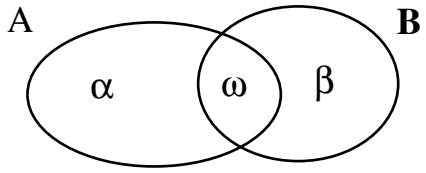


Fig.C.1. L'exemple de deux ensembles avec les cardinalités correspondantes.

Prenons les notations suivantes (cf. Fig.C.1):

$|A - B| = \alpha$,

$|B - A| = \beta$,

$|A \cap B| = \omega$.

Si maintenant nous considérons sur le treillis $\langle L_C, \ll \rangle$, nous pouvons calculer les distances sur le graphe en fonction des trois valeurs citées plus haut. Rappelons que les deux longueurs qui nous intéressent sont sur les chemins reliant les nœuds des deux ensembles avec le nœud correspondant de leur généralisation la plus spécifique. Une fois ces longueurs exprimées, nous aurons trouvé une expression de la fonction δ' qui doit être, *a priori*, égale à $\alpha + \beta$. Plus précisément, notre but sera de montrer que la distance sur le graphe du treillis entre le

noeud correspondant à l'ensemble A et celui correspondant à $A \cap B$ est égale à α. Pour cela, nous allons compter les nœuds intermédiaires entre les deux noeuds fixés, c'est-à-dire ceux qui sont supérieurs au premier et inférieurs ou égaux au deuxième. Tout d'abord, rappelons que $\langle L_C, \ll \rangle$ est isomorphe à $\langle 2^D, \supseteq \rangle$. A cause de l'isomorphisme d'ordres, nous pouvons raisonner sur la deuxième structure. La question devient donc la suivante: combien y a-t-il d'ensembles sur D qui sont intermédiaires pour l'inclusion? Notre affirmation est qu'il en a exactement α. Afin de rendre la preuve de ce fait complètement formelle, une induction très rudimentaire sur α peut être faite. Nous l'évitons à cause de sa simplicité.

Nous nous contenterons de donner un exemple. Soient $D \subseteq$ N avec $A = \{1, 4, 6\}$ et $B = \{3, 4, 8, 16\}$. Nous en déduisons $A \cap B = \{4\}$, $\alpha = 2$ et $\beta = 3$. Les deux chemins correspondants dans le graphe de $\langle 2^D, \supseteq \rangle$ sont donnés sur Fig.C.2.

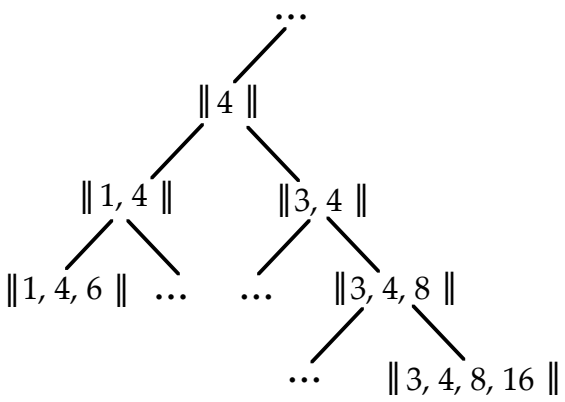


Fig.C.2. L'exemple de deux ensembles avec les cardinalités correspondantes.

La valeur calculée de la fonction normalisée est donc: $\delta'_n(A,B)=\frac{5}{|D|}$. Notons aussi que $A \Delta B = \{1, 3, 6, 8, 16\}$ et donc $|A \Delta B| = 5$.

❑

Lemme 8

δ'_{max} est la cardinalité du domaine, $|D|$.

Preuve:

Il suffit de montrer que: primo, $\delta'_{max} \leq |D|$, secundo, il existe deux ensembles A et B pour lesquels la valeur de δ' atteint le maximum, c'est-à-dire $\delta'(A,B) = |D|$. Pour la première affirmation, il est clair que $A, B \in 2^D$: $(A \Delta B) \subseteq D$. Par conséquent, $\delta'(A,B) = |A \Delta B| \leq |D|$. On pourra donc affirmer $\delta'_{max} = \max\{\delta'(A,B) \mid A, B \in 2^D\} \leq |D|$ (1). Ensuite, soient $A, B \in 2^D$ tels que $B = D - A$. L'ensemble $A \Delta B$ est clairement égal à $A \cup B$, ce qui revient à dire que $A \Delta B = D$. Du coup, $\delta'(A,B) = |A \Delta B| = |D|$. D'où $|D| = \delta'(A,B) \leq \delta'_{max}$ (2). Finalement, d'après (1) et (2) on a $\delta'_{max} = |D|$.

❑

Théorème 9

δ'_n est une distance.

Preuve:

Tout d'abord, le Théorème 1 dans § 3.2. du Chapitre III nous assure que la *fonction d'écart* est une mesure de dissimilarité valide. Afin de prouver qu'elle est de plus une distance, il suffit de montrer qu'elle vérifie l'inégalité du triangle.

Soient A,B et C trois ensembles quelconques. Prenons les notations suivantes (cf. Fig.C.3).

Tout d'abord, soit

$|(A - B) - C| = a$,
$|(B - C) - A| = b$,
$|(C - A) - B| = c$.

Ensuite, soit

$|A \cap B - C| = \gamma$,

$|A \cap C - B| = \beta$,

$|C \cap B - A| = \alpha$.

Finalement, soit $|A \cap B \cap C| = \omega$. Notre but sera de prouver que δ'_n vérifie l'inégalité triangulaire, c'est-à-dire $\delta'_n(A,B) \leq \delta'_n(B,C) + \delta'_n(C,A)$. Pour ce faire il suffira de prouver que $|A \Delta B| \leq |B \Delta C| + |C \Delta A|$. Maintenant, remarquons que:

$|A \Delta B| = a + \beta + b + \alpha$,

$|B \Delta C| = b + \gamma + c + \beta$,

$|C \Delta A| = c + \alpha + a + \gamma$.

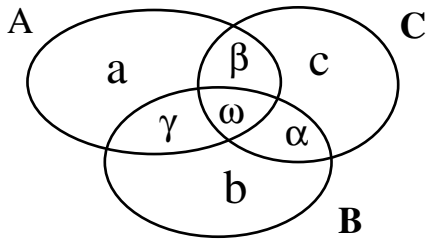


Fig.C.3. L'exemple de trois ensembles avec les cardinalités correspondantes.

Nous avons donc à montrer que $a + \beta + b + \alpha \leq (b + \gamma + c + \beta) + (c + \alpha + a + \gamma)$. Mais après simplification on obtient $0 \leq 2c + 2\gamma$. Ceci est trivialement vrai, le théorème est donc prouvé.

❑

Ppropriétés des produits d'ordres pariels

Dans les prochaines preuves nous utilisons des notions qui relèvent du produit d'ensembles partiellement ordonnés. Pour cette raison nous commençons par citer certains résultats globalement valides sur ce type de structures. Les résultats en question seront largement référencés dans la suite.

Définition 28

Soit $\langle E, \leq \rangle$ une structure d'ordre partiel et e_1, e_2 deux éléments dans E tels que $e_1 \leq e_2$. Une suite $S = \{e_i\}_{i=1,n}$ strictement croissante d'éléments dans E sera appelée *maximale* si:

$\forall\ i,\ 1 \leq i \leq n\text{-}1,\ \neg\exists\ e \in E$ tel que $e_i \prec e$ et $e \prec e_{i+1}$ ($\prec$ est l'ordre stricte induit par $\leq$ sur E).

Soient $\langle E_1, \leq_1 \rangle$ et $\langle E_2, \leq_2 \rangle$ deux structures d'ordre partiel. Soit $\langle E_1 \times E_2, \leq_x \rangle$ la structure produit. Rappelons que pour deux éléments quelconques (e_1,e_2) et (e'_1,e'_2) du produit $E_1 \times E_2$, $(e_1,e_2) \leq_x (e'_1,e'_2)$ ssi $e_1 \leq_1 e'_1$ et $e_2 \leq_2 e'_2$. Il faut remarquer le fait suivant:

Proposition 10:

Pour une suite S dans $E_1 \times E_2$, strictement croissante et maximale si (e_1,e_2) et (e'_1,e'_2) sont deux éléments consécutifs de S, alors exactement une des deux proposition est valide:

$$e_1 = e'_1 \text{ ou } e_2 = e'_2.$$

Preuve:

Par absurde, la négation de la propriété entraîne une contradiction avec la propriété de la maximalité de S.

❑

Notre prochain but sera donc de prouver le résultat suivant rédigé sous forme d'un lemme auxiliaire.

Lemme 11

Il existe une suite S_x entre (e_1,e_2) et (e'_1,e'_2) dans $\langle E_1 \times E_2, \leq_x \rangle$ strictement croissante maximale de longueur k ssi il existe des suites strictement croissantes et maximales S_1 entre e_1 et e'_1 dans E_1 et S_2 entre e_1 et e'_1 dans E_2 respectivement, dont les longueurs respectives, k_1 et k_2 sont telles que $k = k_1 + k_2$.

Preuve:

Condition nécessaire: Il suffit de considérer les deux projection de la suite S_x sur E_1 et E_2. Celles-ci forment deux suites non strictement croissantes, soit S'_x et S''_x. On en soustrait les sous-suites maximales strictement croissantes (autrement dit on ne prend que les éléments différents). Ces dernières sont les deux suites recherchées, S_1 et S_2. La preuve de l'égalité des longueurs utilise le fait qu'entre deux éléments de la suite exactement un nouvel élément, qu'il soit dans S'_x ou dans S''_x, est introduit (ceci est une conséquence de la propriété d'en haut). Le nombre total des nouveaux éléments, dans S'_x et S''_x, s'élève donc au nombre d'éléments de S_x. D'où l'égalité recherchée.

Condition nécessaire: Etant données les deux suites S_1 et S_2, il suffit de les "enchaîner", d'abord, par exemple, S_1 et puisS_2 (sur la façon d'enchaîner deux suites croissantes à l'intérieur d'un produit le lecteur est invité à se rapporter à la preuve du Lemme 9). La suite ainsi obtenue est une suite strictement croissante maximale car les deux composantes le sont et elle a pour longueur la somme des longueurs.

❑

Corollaire 11.1:

Considérons les suites maximales de longueur minimale entre deux éléments. Nous affirmons que S_x est de longueur minimale ssi S_1 et S_2 le sont.

Preuve:

Toutes les deux parties de la proposition admettent une preuve par absurde.

❑

Inersection plus contraintes de longueur

Théorème 12

$\delta'' = \delta' + \delta_{\Im}$. Chacune des deux composantes est calculée sur la partie correspondante de l'expression en L_C.

Preuve:

Rappelons que δ'' est la *dissimilarité topologique* sur $\langle L_C, \ll \rangle$. Et que L_C est le produit Cartésien de deux structures partiellement ordonnées. D'autre part, un chemin dans le graphe d'une telle structure défini une suite strictement croissante selon l'ordre sous-jacent. Les éléments de cette suite correspondent aux nœuds du chemin. Leur nombre est égal, ceci est clair, à la longueur du chemin. Afin d'établir le lien inverse, entre une suite et un chemin, nous utilisons la notion de suite maximale.

Il est clair qu'une bijection entre les suites maximales dans une structure à ordre partiel et les chemins dans le graphe correspondant existe. Désormais nous allons parler de suite entre deux éléments e_1,e_2 si e_1 est le premier élément de la suite et e_2 - le dernier.

La correspondance entre les chemins dans un graphe et les suites maximales nous permet d'affirmer:

Assertion 12"

La longueur du plus court chemin dans le graphe d'un produit de deux ordres partiels est égale à la somme des longueurs des deux plus courts chemins correspondants.

Compte tenu du résultat énoncés ci-dessus (Assertion 12") et en se rappelant que les deux dissimilarités mesurent des longueurs de plus courts chemins, nous avons la certitude que l'égalité suivante est valide

$$\delta'' = \delta' + \delta_{\Im}. \qquad (12.1)$$

❑

Théorème 13

δ'' et δ''_n sont des distances.

Preuve:

D'après Théorème 6, $\delta_{\Im}$ est une distance. Depuis Théorème 9 δ' l'est aussi. La somme de deux distances est aussi une distance. δ'' est donc une distance, ainsi que $2*\delta''_n$. Ensuite, le produit d'une distance et un nombre réel et encore une distance (la preuve sur l'inégalité du triangle). Du coup, δ''_n est une distance.

❑

Assertion 14

Le nombre des combinaisons *redondantes* est exactement

$$\sum_{i=1}^{n} (n - i + 1) \sum_{j=i+1}^{n} \binom{n}{j} .$$

Preuve:

Notre approche consiste, à découper l'ensemble de toutes les valeurs, à indexer les sous-ensembles, calculer leurs cardinalités, et puis les cumuler dans la somme totale. Pour ce faire nous utilisons comme index la borne inférieure de l'intervalle dans $\Im$. Celle-ci parcourt $[1,n]$, il y a donc n nombres à considérer. Pour chacun d'entre eux, soit i, nous avons besoin de connaître deux quantités. La première reflète le nombre des intervalles ayant pour borne inférieure i. La deuxième est le nombre d'ensembles sur D dont la cardinalité est strictement supérieure à i.

Pour la première valeur, un simple calcul nous permet d'affirmer qu'il y a $n - i + 1$ tels intervalles. Le calcul de la deuxième nécessite des résultats de combinatoire, et notamment le fait qu'il y a $\binom{n}{k}$ ensembles différents de cardinalité k sur D. Pour un i donné donc il y a exactement $\sum_{j=i+1}^{n} \binom{n}{j}$ ensembles différents de cardinalité supérieure à i. Maintenant si nous prenons en compte que chaque couple (ensemble, intervalle) présente une combinaison différente, le résultat recherché pour i devient:

$$(n - i + 1) \sum_{j=i+1}^{n} \binom{n}{j} .$$

Après un cumul sur i nous obtenons que la cardinalité totale de Θ est

$$\sum_{i=1}^{n} (n - i + 1) \sum_{j=i+1}^{n} \binom{n}{j} .$$

❑

Lemme 15

L'ensemble des éléments *admissibles* est fermé par généralisation. Autrement exprimé

$\forall A, B \in 2^D \times \Im$, si A et $B \in \Omega$, alors Gen*$(A,B) \in \Omega$.

Preuve:

Pour la preuve, il suffit de considérer la façon dont la valeur de Gen* (A,B) est calculée. Supposons que $A = E_1 \times [i_1,j_1]$ et $B = E_2 \times [i_2,j_2]$ où $E_1, E_2 \in 2^D$ et $[i_1,j_1], [i_2,j_2] \in \Im$. D'après la définition de la généralisation, Gen*$(A,B) = E_1 \cap E_2 \times [\min(i_1,i_2), \max(j_1,j_2)]$. Nous savons encore, de par l'énoncé du lemme, que $|E_1| \leq i_1$ et $|E_2| \leq i_2$. Il reste à montrer que le même rapport entre $|E_1 \cap E_2|$ et $\min(i_1,i_2)$. D'abord, il est clair que $E_1 \cap E_2 \subseteq E_1$ et donc

$$|E_1 \cap E_2| \leq |E_1| \leq i_1. \qquad (15.1)$$

Du même

$$|E_1 \cap E_2| \leq |E_2| \leq i_2. \qquad (15.2)$$

La prise en compte simultanée de (15.1) et (15.2) nous permet de conclure que

$$|E_1 \cap E_2| \leq \min(i_1,i_2). \qquad (15.3)$$

La dernière inégalité montre que Gen*$(A,B) \in \Omega$.

❑

Lemme 16

Entre deux sommets comparables *admissibles*, il y a toujours un chemin à longueur minimale (calculée selon la formule (12.1) plus haut) *admissible*.

Preuve:

La preuve est basée essentiellement sur la correspondance entre les suites maximales d'éléments et les chemins dans le graphe d'une structure. En gros, l'idée consiste à prendre un chemin de longueur minimale et à le retravailler afin d'en obtenir un autre dont on est sûr qu'il est admissible. Pendant le processus

Soient $A, B \in 2^D \times \Im$ tels que $A,B \in \Omega$ et $A \leq B$ et soit σ un chemin de longueur minimale entre A et B dans le graphe de $2^D \times \Im$. Prenons la suite d'éléments S correspondante à σ. Nous avons déjà indiqué qu'une telle suite est strictement croissante maximale. De plus, il est possible d'en extraire deux suites strictement croissantes maximales, chacune sur une des dimensions du produit (cf. le Lemme 11). Appelons ces suites S_1 et S_2 obtenues respectivement sur 2^D et $\Im$. Dans la suite de cette preuve nous allons montrer comment "construire" un chemin *admissible* à partir de S_1 et S_2.

Tout d'abord, rappelons qu'un élément *redondant* de $2^D \times \Im$ se caractérise par une supériorité stricte de la cardinalité de l'ensemble dans 2^D par rapport à la borne inférieure de l'intervalle dans $\Im$. Ensuite, notons le fait que pour une suite (strictement) croissante S_1 dans 2^Dla suite des cardinalités correspondantes S_c est une suite (strictement) décroissante dans N. La même assertion peut être fait sur une suite (strictement) croissante S_2 dans $\Im$ et la suite des bornes inférieures correspondantes S_{bi}, avec la remarque que cette dernière n'est pas strictement décroissante même si la croissance de S_2 est stricte. Nous sommes donc en présence de deux suites décroissantes de nombres entiers, S_c et S_{bi}. Une façon particulière de composer ces suites est reflétée par la structure de la suite S. Plus précisément, si l'on imagine la suite S_c (pour global) de paires (carinalité, borne inférieure° mesurées sur les éléments de S, alors S_c et S_{bi} sont les composantes de S_c en même titre que S_1 et S_2 le sont pour S. De plus, nous pouvons traduire la présence d'un élément redondant dans S par une inégalité entre les occurrences des éléments correspondantes de S_c et S_{bi}. L'absence de tels éléments peut, elle aussi, se traduire par une condition sur la façon de composer ces deux suites. Il ne nous reste donc qu'à énoncer cette condition puis à montrer une composition possible.

En ce qui concerne la condition, elle est plutôt intuitive. En effet, nous allons exiger que la suite S_c "décroisse plus rapidement" que S_{bi} à l'intérieur de S_c. Ceci se reporte sur les suites d'éléments S_1 et S_2 dont la première devrait croître plus rapidement que la seconde dans leur composition. Nous proposons une façon de satisfaire la condition énoncée, il consiste en effet à coller les deux suites, d'abord S_1 et puis S_2. Plus formellement, soient les deux suites

$$S_1 = \{E_i\}_{i=1,k} \text{ et}$$
$$S_2 = \{[p_j,q_j]\}_{j=1,m}.$$

Nous savons que $A = E_1 \times [p_1,q_1]$ et $B = E_k \times [p_m,q_m]$, et que

$$|E_1| \leq p_1 \quad (A \in \Omega) \qquad (16.1)$$

et

$$|E_k| \leq p_m \quad (B \in \Omega). \qquad (16.2)$$

Si maintenant nous construisons une composition S' de S_1 et S_2 de la façon suivante:

$$S' = \{C_r\}_{r=1,k+m} \ (C_r = E_r \times [p_r,q_r]\ ).$$

et

$$C_r = E_r \times [p_1,q_1] \text{ pour } r=1,k \text{ et } C_r = E_k \times [p_{r-k+1},q_{r-k+1}] \text{ pour } r=k+1,k+m,$$

alors cette composition respecte la condition. Ceci devrait être démontré séparément pour les deux parties de S' en se basant sur les propriétés de décroissance des suites S_c et S_{bi}. Pour la première partie de $\{C_r\}$ avec $r=1,k$ nous pouvons observer que la cardinalité de l'ensemble parcourt la suite S_c tandis que la borne inférieure de l'intervalle reste toujours égale à p_1. La condition est vérifiée sur l'intégralité de cette partie de S' car:

(i) elle est vérifiée sur le premier éléments, égal à A , d'après (16.1),

(ii) la suite S_c est strictement décroissante, c'est-à-dire

$$\forall\ r,\ 1 < r \leq k,\ |E_r| < |E_1|. \qquad (16.3)$$

Du coup, nous pouvons combiner les deux résultats en

$$\forall\ r,\ 1 < r \leq k,\ |E_r| < p_1 \qquad (16.4)$$

ce que revient à affirmer

$$\forall\ r,\ 1 < r \leq k,\ |E_r| < p_r. \qquad (16.4')$$

Il reste à prouver le même résultat sur la partie complémentaire de S'. La démonstration suit essentiellement le schéma d'en haut. Elle est symétrique à la précédente, dans la mesure où nous utilisons le résultat (16.2) qui porte non pas sur le premier élément de S', mais sur le dernier, $E_{k+m} \times [p_{k+m},q_{k+m}] = E_k \times [p_m,q_m] = B$. De même nous utilisons le fait que si le dernier élément (en l'occurrence $p_{k+m} = p_m$) d'une suite décroissante (S_{bi}) est supérieur à une valeur ($|E_k|$), alors la totalité des éléments de la suite sont aussi supérieures. Concrètement, puisque

$$\forall\ r,\ k+1 \leq r \leq k+m,\ p_m \leq p_r \qquad (16.5)$$

et d'après (16.2) nous pouvons conclure que

$$\forall\ r,\ k+1 \leq r \leq k+m,\ |E_r| = |E_k| \leq p_r. \qquad (16.6)$$

Ainsi se termine-t-elle la deuxième partie de notre preuve. Finalement, en combinant les assertions (16.4') et (16.6) nous obtenons pour la suite $S' = \{C_r\}_{r=1,k+m}$ $(C_r = E_r \times [p_r,q_r]\ )$

$$\forall\ r,\ 1 \leq r \leq k+m,\ |E_r| < p_r. \qquad (16.7)$$

La dernière inégalité nous assure que S' respecte la condition examinée, c'est-à-dire que tous ses éléments sont dans Ω. De plus, c'est une suite entre A et B, strictement croissante (par construction) et de longueur $k+m$, donc minimale (car égale à la longueur de la suite de départ S). Il reste à prouver que c'est une suite maximale afin d'avoir le droit d'en extraire le chemin dans le graphe de $2^D \times \Im$ correspondant.

Pour ce fait nous donnons une preuve rapide par l'absurde. Si l'on admet que la suite S' n'est pas minimale, alors il existerait un i avec $1 \leq i \leq k+m-1$ et un élément $F \in 2^D \times \Im$ tels que $C_i < F < C_{i+1}$. On en déduit que au moins sur une des dimensions la projection de F est différente de celles de C_i et C_{i+1} en même temps (sinon il aurait pas les deux inégalités strictes). La suite

$$S^* = \{C_r{}^*\}_{r=1,k+m+1} \text{ avec } C_r{}^* = C_r \text{ pour } 1 \leq r \leq i \text{ et } i+2 \leq r \leq k+m+1 \text{ et } C_{i+1}{}^* = F$$

est une suite strictement croissante. On en extrait les suites $S_1{}^*$ et $S_2{}^*$ de la même façon que les suites S_1 et S_2 en haut. Ceci signifie, en particulier, que les deux suites obtenues sont strictement croissantes. Il est clair qu'au moins une des deux nouvelles suites contient la suite initiale correspondante (S_1 ou S_2) en sous-suite propre. Mais ceci va à l'encontre de la propriété de maximalité des deux suites de départ. La contradiction à laquelle nous aboutissons nous permet d'exclure l'existence de i et F. La suite S' est donc maximale.

La totalité des résultats ci-dessus nous donne la certitude qu'en choisissant les noeuds correspondant aux éléments consécutifs de la suite nous obtiendrons un chemin entre A et B qui sera:

(i) *admissible*,

(ii) de longueur minimale.
Ceci termine la preuve.

❑

Variante intersection/union

Lemme 17

δ''' peut être calculée en somme de δ' et δ_τ, c'est-à-dire que

$$\delta''' = \delta' + \delta_\tau.$$

Preuve:
Ceci est une conséquence du Lemme sur les séquences strictement croissantes maximales dans un produit (Lemme 11, §). Car comme dans le cas précédent, il s'agit de mesurer des longueurs de chemins dans le graphe du treillis dont l'ensemble est en bijection avec les suites de ce type.

❑

Lemme 18

δ'''_{max} est le double de la cardinalité du domaine, $2*|D|$.

Preuve:
Nous utilisons le même principe que dans la preuve du Lemme 8. Premièrement, dans le même Lemme nous avons prouvé que

$$\delta'_{max} = |D|. \qquad (18.1)$$

Nous avons donc

$$\delta' \leq |D|. \qquad (18.2)$$

A cause de l'égalité

$$\delta_\tau \leq |D|. \qquad (18.3)$$

En ajoutant les deux dernières inégalités nous obtenons

$$\delta''' = \delta' + \delta_\tau \leq 2*|D|. \qquad (18.4)$$

Par conséquent

$$\delta'''_{max} \leq 2*|D|. \qquad (18.5)$$

De même, on peut prendre les mêmes ensembles $A, B \in 2^D$ tels que $B = D - A$. Comme nous l'avons indiqué plus haut, $\delta'(A,B) = |A \Delta B| = |D|$, ainsi que $\delta_\tau(A,B) = |A \Delta B| = |D|$. Du coup, $\delta'''(A,B) = 2*|D|$ et donc on peut affirmer que

$$\delta'''_{max} \geq 2*|D|. \qquad (18.6)$$

De (18.6) et (18.5) on déduit

$$\delta'''_{max} = 2*|D|. \qquad (18.6)$$

❑

Théorème 19

δ''' et δ'''_n sont des distances.

Preuve:

La première dissimilarité est la somme de deux distances. L'opérateur d'addition préserve les propriétés d'une distance, elle l'est donc. La deuxième est le produit d'une distance avec un nombre réel, positif. Cette opération préserve aussi, ceci n'est pas difficile à prouver, la qualité d'être une distance.

❑

Assertion 20

Soit $|D| = n$. Le nombre des combinaisons *absurdes* est exactement

$$\sum_{k=0}^{n} \binom{n}{k} * (2^{|D|} - 2^{k}).$$

Preuve:

Nous utilisons une approche semblable à celle de la preuve de l'Assertion 14. Le découpage et l'indexation se feront sur les ensembles dans la partie union de $2^D \times 2^D$. Pour un ensemble union donné nous allons calculer le nombre des ensembles intersections incompatibles. Ensuite nous allons faire le cumul. Une telle démarche est correcte, car, ici encore, les combinaisons entre deux éléments sont uniquement déterminées par ces éléments (on ne risque pas de compter deux fois).

Pour la première partie, le calcul des ensembles intersections incompatibles, nous prenons un ensemble quelconque $A \in 2^D$. Soit $|A| = k$. Le nombre des ensembles de 2^D incompatibles avec A peut en effet être calculé de manière indirecte, en soustrayant de $2^{|D|}$ le nombre des ensembles compatibles, c'est-à-dire ceux qui sont compris dans A. Leur nombre est, fait connu, 2^k. L'apport de A dans la cardinalité de Ξ s'élève donc à $2^{|D|} - 2^k$. Il faut maintenant cumuler cette valeur sur:

(i) le nombre total de tels A, c'est-à-dire à cardinalité k (car le seul fait que nous avons exploité pour le calcul ci-dessus est la cardinalité fixée). Le nombre de ces ensembles est $\binom{n}{k}$.

(ii) le nombre total de différentes cardinalités, n.

Ainsi, la somme finale devient

$$\sum_{k=0}^{n} \binom{n}{k} * (2^{|D|} - 2^{k}). \qquad (20.1)$$

❑

Lemme 21

Ω est fermé par généralisation. En outre,

$\forall A, B \in 2^D \times 2^D$, si A et $B \in \Omega$, alors Gen*$(A,B) \in \Omega$.

Preuve:

Soit $A = E_1 \times E_1'$ et $B = E_2 \times E_2'$. D'après l'admissibilité de A et B nous savons que

$$E_1 \subseteq E_1' \qquad (21.1)$$

et

$$E_2 \subseteq E_2'. \qquad (21.2)$$

De l'autre côté, Gen* $(A,B) = \|E_1 \cap E_2\| \times \{E_1' \cup E_2'\}$. Notre but sera donc à prouver que $(E_1 \cap E_2) \subseteq (E_1' \cup E_2')$. Ceci est immédiat, car

$$(E_1 \cap E_2) \subseteq E_2 \qquad (21.3)$$

et puis

$$E_2' \subseteq (E_1' \cup E_2'). \qquad (21.4)$$

Il suffit maintenant de combiner (21.3), (21.2) et (21.3) pour obtenir

$$(E_1 \cap E_2) \subseteq E_2 \subseteq E_2' \subseteq (E_1' \cup E_2'). \qquad (21.5)$$

Ceci termine la preuve de Gen*$(A,B) \in \Omega$.

❑

Lemme 22

Entre deux sommets comparables *admissibles*, il y a toujours un chemin de longueur minimale *admissible*.

Preuve:

La preuve peut être construite selon le même principe que celle du Lemme 16. A partir d'un chemin de longueur minimale entre deux sommets nous construisons un autre (en général) chemin. Le deuxième chemin est admissible et il a la même longueur.

Nous nous limitons sur l'indication des éléments différents par rapport à la preuve du Lemme 16. Ici nous utilisons directement des suites d'ensembles, ordonnées pour l'inclusion, extraites à partir de la suite correspondante au chemin du départ. A noter que les ordres sont inversés entre la suite des intersections et celle des unions. Le raisonnement est plus simple dans la mesure où les cardinalités des ensembles sont laissées de coté. Finalement, dans la construction du chemin définitif, comme avant, les résultats formulés par Lemme 16 sont utilisés.

❑

Annexe D

Un exemple d'application de l'algorithme CLUSTER/SC.

La description d'une base dans le syntaxe de TROPES est donnée. La base, nommée `real-estate` contient un seule concept, `house`. Les quatorze instances du concept sont caractérisées par neuf attributs: `address`, `number`, `floor`, `surface`, etc. Un seul point de vue est défini sur le concept en considération: `test`.

```
<real-estate
  bases-requises = {};
  types = {};
  points-de-vue = { test } ;
  concepts = {
      <house
        clefs = {
            <address  dans = chaine ; nature = propriete ; const = un ;>,
            <number   dans = entier ; nature = propriete ; const = un ;>,
            <floor    dans = entier ; nature = propriete ; const = un ;>
                  } ;
        attributs = {
            <surface  dans = entier ; nature = propriete ; const = un ;>,
            <nb-rooms dans = entier ; nature = propriete ; const = un ;>,
            <agent    dans = chaine ; nature = propriete ; const = un ;>,
            <price    dans = entier ; nature = propriete ; const = un ;>,
            <area1    dans = entier ; nature = propriete ; const = un ;>,
            <area2    dans = entier ; nature = propriete ; const = un ;>
            };
        points-de-vue = {
            <test
              classe-racine =
                <house
                  attributs = {
                    <address  facettes = { domaine = { "meylan", "eybens",
"grenoble", "smh", "fontaine" }; }; >,
                    <number   facettes = { intervalles = {[1 200]} ; };>,
                    <floor    facettes = { intervalles = {[0 10]} ; }; >,
                    <area1    facettes = { intervalles = {[1 5]} ; }; >,
                    <area2    facettes = { intervalles = {[1 5]}; }; >,
                    <price    facettes = { intervalles = {[20 100]}; };>,
                    <surface  facettes = { intervalles = {[20 100]}; };>,
                    <nb-rooms facettes = { intervalles = {[1 5]}; };>,
                    <agent    facettes = { domaine = { "alpimmo", "ig",
"unimmo", "eid" }; };>
                                        };
                >;
              classes = {};
            >
                          };

        passerelles = {};

      instances = {
```

```
< est-dans = { house test } ;
  address = "fontaine";
  number =  57 ;
  floor = 5 ;
  surface = 48 ;
  nb-rooms = 2;
  price = 52 ;
  area1 = 2 ;
  area2 = 4 ;
      agent = "eid";
>,
< est-dans = { house test } ;
  address = "smh";
  number =  29 ;
  floor = 7 ;
  surface = 65 ;
  nb-rooms = 3;
  price = 60 ;
  area1 = 3 ;
  area2 = 5 ;
      agent = "alpimmo";
>,
< est-dans = { house test } ;
  address = "meylan";
  number = 69 ;
  floor = 4 ;
  surface = 56 ;
  nb-rooms = 2;
  price = 68 ;
  area1 = 1 ;
  area2 = 4 ;
      agent = "eid";
>,
< est-dans = { house test } ;
  address = "smh";
  number =  13 ;
  floor = 5 ;
  surface = 73 ;
  nb-rooms = 4;
  price = 82 ;
  area1 = 2 ;
  area2 = 5 ;
      agent = "unimmo";
>,
< est-dans = { house test } ;
  address = "meylan";
  number =  18 ;
  floor = 9 ;
  surface = 41 ;
  nb-rooms = 2;
  price = 60 ;
  area1 = 1 ;
  area2 = 2 ;
      agent = "alpimmo";
>,
< est-dans = { house test } ;
  address = "grenoble";
  number =  33 ;
  floor = 1 ;
  surface = 58 ;
  nb-rooms = 3;
  price = 86 ;
```

```
  area1 = 4 ;
  area2 = 2 ;
      agent = "eid";
>,
< est-dans = { house test } ;
  address = "meylan";
  number =  176 ;
  floor = 3 ;
  surface = 35 ;
  nb-rooms = 1;
  price = 31 ;
  area1 = 3 ;
  area2 = 1 ;
      agent = "alpimmo";
>,
< est-dans = { house test } ;
  address = "smh";
  number =  95 ;
  floor = 3 ;
  surface = 20 ;
  nb-rooms = 1;
  price = 20 ;
  area1 = 2 ;
  area2 = 3 ;
      agent = "ig" ;
>,
< est-dans = { house test } ;
  address = "grenoble";
  number =  113 ;
  floor = 0 ;
  surface = 60 ;
  nb-rooms = 2;
  price = 35 ;
  area1 = 2 ;
  area2 = 2 ;
      agent = "unimmo" ;
>,
< est-dans = { house test } ;
  address = "meylan";
  number =  60 ;
  floor = 6 ;
  surface = 74 ;
  nb-rooms = 3;
  price = 80 ;
  area1 = 3 ;
  area2 = 1 ;
      agent = "eid" ;
>,
< est-dans = { house test } ;
  address = "eybens";
  number =  47 ;
  floor = 0 ;
  surface = 80 ;
  nb-rooms = 3;
  price = 75 ;
  area1 = 1 ;
  area2 = 2 ;
      agent = "ig" ;
>,
< est-dans = { house test } ;
  address = "eybens";
  number =  13 ;
```

```
              floor = 4 ;
              surface = 65 ;
              nb-rooms = 3;
              price = 70 ;
              area1 = 2 ;
              area2 = 1 ;
                  agent = "ig" ;
            >,
            < est-dans = { house test } ;
              address = "grenoble";
              number =  21 ;
              floor = 2 ;
              surface = 70 ;
              nb-rooms = 3;
              price = 85 ;
              area1 = 3 ;
              area2 = 2 ;
                  agent = "unimmo" ;
            >,
            < est-dans = { house test } ;
              address = "grenoble";
              number =  32 ;
              floor = 3 ;
              surface = 90 ;
              nb-rooms = 4;
              price = 100 ;
              area1 = 1 ;
              area2 = 2 ;
                  agent = "eid" ;
            >
                 };
      >
  };
  fichiers-instances = {};
  modules-talk = {};
>
```

Les quatorze instances du concept ont été regroupées par l'algorithme dans trois classes que nous donnons ci-dessous dans le format interne de TROPES. La liste des instances d'une classe est définie par l'attribut `object-list` (à noter que les individus sont représentés par leur numéros et que ceux-ci sont attribués dans l'ordre inverse et en commençant par *0*) tandis que sa description est introduite par `l-complex`.

```
#s(.class rank: 445 object-list: (7 12 13 9 5 10 6 2)

l-complex: (

#s(.selector rank: 8239 slotname: "number" relation: eq val-list: ((tr
-make-interval 13 t 13 t) (tr-make-interval 18 t 18 t) (tr-make-interval 29
t 29 t) (tr-make-interval 57 t 57 t) (tr-make-interval 95 t 95 t) (tr-make-
interval 113 t 113 t) (tr-make-interval 176 t 176 t)))

#s(.selector rank: 8240 slotname: "floor" relation: eq val-list: ((tr-make-
interval 0 t 0 t) (tr-make-interval 3 t 9 t)))

#s(.selector rank: 8241 slotname: "surface" relation: eq val-list: ((tr-
make-interval 20 t 20 t) (tr-make-interval 35 t 35 t) (tr-make-interval 41
t 41 t) (tr-make-interval 48 t 48 t) (tr-make-interval 60 t 60 t) (tr-make-
interval 65 t 65 t) (tr-make-interval 73 t 73 t)))
```

```
#s(.selector rank: 8242 slotname: "nb-rooms" relation: eq val-list: ((tr-
make-interval 1 t 4 t)))

#s(.selector rank: 8243 slotname: "price" relation: eq val-list: ((tr-make-
interval 20 t 20 t) (tr-make-interval 31 t 31 t) (tr-make-interval 35 t 35
t) (tr-make-interval 52 t 52 t) (tr-make-interval 60 t 60 t) (tr-make-
interval 70 t 70 t) (tr-make-interval 82 t 82 t)))

#s(.selector rank: 8244 slotname: "area1" relation: eq val-list: ((tr-make-
interval 1 t 3 t)))))



#s(.class rank: 446 object-list: (11 4)

l-complex: (

#s(.selector rank: 8246 slotname: "address" relation: eq val-list: ("smh"
"meylan"))

#s(.selector rank: 8247 slotname: "number" relation: eq val-list: ((tr-
make-interval 60 t 60 t) (tr-make-interval 69 t 69 t)))

#s(.selector rank: 8248 slotname: "floor" relation: eq val-list: ((tr-make-
interval 4 t 6 t)))

#s(.selector rank: 8249 slotname: "surface" relation: eq val-list: ((tr-
make-interval 56 t 56 t) (tr-make-interval 74 t 74 t)))

#s(.selector rank: 8250 slotname: "nb-rooms" relation: eq val-list: ((tr-
make-interval 1 t 3 t)))

#s(.selector rank: 8251 slotname: "agent" relation: eq val-list: ("eid"))

#s(.selector rank: 8252 slotname: "price" relation: eq val-list: ((tr-make-
interval 68 t 68 t) (tr-make-interval 80 t 80 t)))

#s(.selector rank: 8253 slotname: "area1" relation: eq val-list: ((tr-make-
interval 1 t 1 t) (tr-make-interval 3 t 3 t)))

#s(.selector rank: 8254 slotname: "area2" relation: eq val-list: ((tr-make-
interval 1 t 1 t) (tr-make-interval 4 t 4 t)))))



#s(.class rank: 447 object-list: (8 3 1 0)

l-complex: (

#s(.selector rank: 8255 slotname: "address" relation: eq val-list:("eybens"
"grenoble"))

#s(.selector rank: 8256 slotname: "number" relation: eq val-list: ((tr-
make-interval 21 t 21 t) (tr-make-interval 32 t 33 t) (tr-make-interval 47
t 47 t)))

#s(.selector rank: 8257 slotname: "floor" relation: eq val-list: ((tr-make-
interval 0 t 3 t)))

#s(.selector rank: 8258 slotname: "surface" relation: eq val-list: ((tr-
make-interval 58 t 58 t) (tr-make-interval 70 t 70 t) (tr-make-interval 80
t 80 t) (tr-make-interval 90 t 90 t)))
```

```
#s(.selector rank: 8259 slotname: "nb-rooms" relation: eq val-list: ((tr-
make-interval 3 t 4 t)))

#s(.selector rank: 8260 slotname: "price" relation: eq val-list: ((tr-make-
interval 75 t 75 t) (tr-make-interval 85 t 86 t) (tr-make-interval 100 t
100 t)))

#s(.selector rank: 8261 slotname: "area1" relation: eq val-list: ((tr-make-
interval 1 t 4 t)))

#s(.selector rank: 8262 slotname: "area2" relation: eq val-list: ((tr-make-
interval 2 t 2 t)))))
```