Semaine de la Connaissances / Nantes, du 26 au 30 juin 2006

DEUXIÈME ATELIER :

Représentation et raisonnement sur le temps et l'espace (RTE 2006)

Florence Le Ber
Gérard Ligozat
Odile Papini

2e atelier « Représentation et Raisonnement sur le Temps et l'Espace »

RTE2006

Dans le cadre de la Semaine de la Connaissance 2006

Le 27 juin 2006 à Nantes (F)
Université de Nantes

Présidente du comité d'organisation : Mounira Harzallah (LINA, Nantes)
Présidents du comité de pilotage : Nathalie Aussenac-Gilles (IRIT, Toulouse) et Jean Charlet (AP-HP, Paris)
Site web de la plate-forme : http://www.sdc2006.org.

Avant propos

La représentation du temps et de l'espace, et les modèles de raisonnements associés, sont des thématiques largement développées en Intelligence Artificielle, mais qui concernent également d'autres domaines de recherche et de nombreuses applications (gestion de l'espace, prévention des risques naturels, etc.). Les thématiques abordées vont des systèmes d'information géographique, et des questions liées, comme la qualité des données, aux formalismes axiomatiques et algébriques de relations spatio-temporelles et aux problèmes de résolution de contraintes. Le traitement du langage naturel et la représentation de connaissances sont également des thèmes connexes.

L'atelier RTE, qui s'est tenu pour la première fois en 2005 à Nice dans le cadre de la plateforme AFIA, a pour objectif de réunir la communauté des chercheurs travaillant dans ce domaine et de discuter des travaux en cours, projets et idées. Il est ouvert à la présentation de travaux de chercheurs et doctorants portant sur l'un ou plusieurs des thèmes suivants : modélisation du temps, modélisation de l'espace, raisonnement spatial qualitatif, raisonnement temporel qualitatif, traitement du langage naturel, représentation de connaissances, applications, etc. Les présentations pourront aborder ces concepts selon différentes problématiques. Un objectif de l'atelier cette année est également de constituer un groupe français pour fédérer et animer la recherche en matière de représentation et de raisonnement sur le temps et l'espace.

Pour cette deuxième manifestation, qui se tient à Nantes dans le cadre de la Semaine de la Connaissance,nous avons recueilli cinq soumissions qui présentent des travaux dans les différents champs cités : représentation d'évènements spatio-temporels, modélisation de relations spatiales, contraintes spatiales ou temporelles, temps et programmation, avec des applications variées : recherche de documents, cartographie, etc. Ces présentations de travaux seront accompagnées d'un exposé synthétique, par Odile Papini, sur le thème de la représentation de l'espace en logique modale.

Nous souhaitons que les participants à cet atelier trouvent matière à idées et discussions dans l'ensemble des présentations et que ces discussions puissent se prolonger lors de futures manifestations. Enfin, nous remercions chaleureusement les membres du comité de programme ainsi que les membres du comité d'organisation de la Semaine de la Connaissance, sans lesquels cet atelier n'aurait pu se tenir.

Florence Le Ber, Gérard Ligozat, Odile Papini

Nantes, le 27 juin 2006

Comité de programme RTE2006

Membres : Maroua Bouzid (GREYC, Caen)
Jean-François Condotta (CRIL, Lens)
Florence Le Ber (CEVH, Strasbourg / LORIA, Nancy)
Gérard Ligozat (LIMSI, Orsay)
Philippe Muller (IRIT, Toulouse)
Odile Papini (LSIS, Toulon)

Relecteurs extérieurs :
Luc Maranget (INRIA, Rocquencourt)

Table des matières

Raisonnement spatial sur la position et l'orientation de solides dans l'espace

Vers une algèbre des badaboums

Olivier Bonin[1], Benoit Poupeau[1]

Institut Géographique National,
Laboratoire COGIT,
2-4 avenue Pasteur, 94165 Saint-Mandé Cedex
{olivier.bonin;benoit.poupeau}@ign.fr

Résumé

Nous présentons dans cet article les premiers travaux sur deux cas de raisonnement semi-qualitatif et qualitatif spatial.

Le premier raisonnement concerne la position relative de solides dans l'espace, exprimée en termes usuels de relations cardinales étendues des notions de *haut* et de *bas*, ce qui donne 27 relations cardinales dans l'espace pour notre modèle.

Nous ajoutons un deuxième raisonnement sur les liens entre solides, dans le cas où ces solides sont en contact direct ou indirect. Nous caractérisons dans le même système de relations cardinales la direction de la normale au plan de contact entre solides. En outre, nous faisons la distinction entre lien majeur (contact direct), lien intermédiaire (contact indirect avec maintien de la direction cardinale des normales des plans de contact), et lien mineur (contact indirect sans maintien des directions des contacts). Nous obtenons ainsi un ensemble de 54 relations pour décrire les liens entre deux solides dans l'espace.

Mots-clés : Relations cardinales 3D, orientation de solides dans l'espace, calcul qualitatif.

1 Introduction

Un nouveau modèle pour représenter et manipuler l'information géographique tri-dimensionnelle a été introduit dans (11). Ce modèle propose un changement de perspective par rapport à la modélisation 3D classique usuelle (7; 12; 9). Plutôt que de se concentrer sur la modélisation géométrique ou géométrico-topologique (par exemple à l'aide de complexes cellulaires ou de complexes simpliciaux) dans le but d'exploiter le maximum d'information des données avec un même modèle, les objets géographiques 3D de ce modèle ont trois représentations (Figure 1) :

- une représentation par frontière, permettant l'affichage et le stockage des données ;
- une représentation par énumération spatiale (pavage irrégulier de tétraèdres) permettant les requêtes topologiques (inclusion par exemple) et volumiques (calcul de volume par exemple) ;
- une représentation que nous appelons cristallographique par un ou plusieurs cristaux appartenant à des habitus des sept systèmes cristallins.

FIG. 1 – Modélisation d'un cube

La représentation cristallographique que nous proposons est particulièrement intéressante, car les cristaux sont des solides dont les propriétés sont bien connues. La propriété la plus caractéristique des cristaux est leur symétrie, représentée par des groupes ponctuels (groupes des roto-inversions et roto-réflexions) (Figure 2). De plus, les faces d'un cristal sont repérées dans l'espace par un système local normalisé appelé *indices de Miller*, et il est aisé de déduire la position relative des faces d'un cristal à l'aide de ces indices. Toutes les faces de même indices de Miller sont parallèles, et ont donc même orientation. Il suffit donc de coupler ces indices de Miller à l'orientation du repère local pour calculer de toutes les faces d'un cristal.

Dans leur application aux données géographiques 3D, nous choisissons des cristaux réduits à des mailles élémentaires (c'est-à-dire qu'il n'y a pas répétition d'un motif : on ne s'intéresse pas aux groupes d'espace). En revanche, ces cristaux peuvent être associés les uns aux autres par deux opérations : la macle, qui associe plusieurs cristaux d'un même système le long d'un plan avec une rotation relative des cristaux, et l'épitaxie, qui associe des cristaux de systèmes différents.

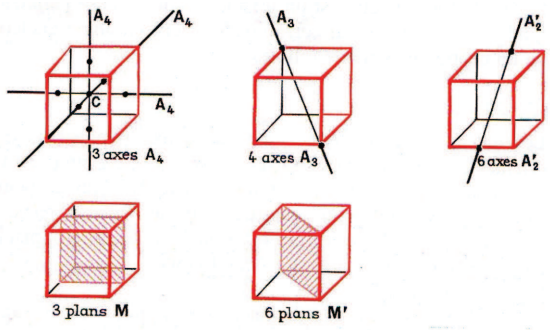


FIG. 2 – Axes de symétrie d'un cube (d'après (10))

Cette description cristallographique des objets géographiques 3D est appliquée aux objets du sol (bâtiments par exemple), et du sous-sol. Un empilement de couches géologiques correspond donc à une superposition de cristaux du modèle (Figure 3).

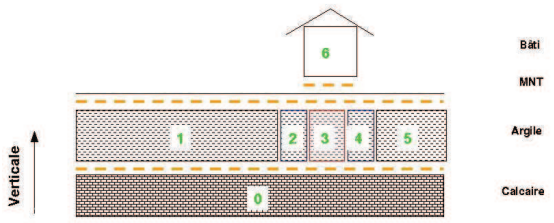


FIG. 3 – Schéma d'un empilement de cristaux représentant un maison et une partie du sous-sol

Nous faisons alors abstraction de la géométrie de ces cristaux, pour raisonner sur les objets géographiques modélisés par ces cristaux au niveau logique, en exploitant les symétries, l'indexation des faces, et les liens entre cristaux. C'est un niveau de description supérieur au niveau géométrico-topologique.

Il paraît donc intéressant de se doter d'un calcul qualitatif sur ces cristaux permettant de répondre à des requêtes géographiques types :

- Quelles sont les couches géologiques sous ce bâtiment ?
- Sont-elles de même pendage ?
- Quel est ce pendage ?
- Quelle zone l'effondrement d'une carrière souterraine est-elle susceptible de toucher ?
- ...

Ces requêtes présentent une forte similarité avec le calcul maintenant classique des relations cardinales (4; 2). Pour pouvoir les satisfaire, nous introduisons dans cet article deux algèbres de relations.

La propriété principale des cristaux étant leur symétrie, nous avons introduit une propriété géographique liée à l'orientation : les directions cardinales. Ces directions font l'objet d'une première algèbre. Nous analysons également les liens entre cristaux à l'aide d'une deuxième algèbre.

La première algèbre constitue une simple extension de l'algèbre des relations cardinales. Les cristaux sont réduits à leur centre de symétrie ou à leur centre de gravité si celui-ci n'existe pas, et on raisonne donc sur des points. Elle donne donc la position relative dans l'espace de deux solides (des cristaux dans notre contexte) en termes de nord N, sud S, est E, ouest O, haut H, bas B, à-niveau L, et de directions intermédiaires latérales, ce qui donne 27 relations possibles : H, H-N, H-S, H-E, H-O, H-NE, H-NO, H-SE, H-SO, L, L-N, L-S, L-E, L-O, L-NE, L-NO, L-SE, L-SO, B, B-N, B-S, B-E, B-O, B-NE, B-NO, B-SE, B-SO. La lettre L seule décrit l'identité des positions relatives, tandis que les lettres H et B seule décrivent une direction dans un petit cône vers le nord et vers le bas. Nous n'introduisons pas de relations intermédiaires verticales. Ces directions sont déterminées à partir des positions des centres de symétrie des cristaux. La première algèbre constitue un calcul semi-qualitatif, ainsi que qualitatif au sens de Ligozat (6) (c'est-à-dire une algèbre semi-associative et une représentation de cette algèbre). Nous qualifions de semi-qualitatif le calcul qui n'utilise pas les compositions entre relations pour effectuer des inférences, mais recalcule les directions cardinales à l'aide de calcul vectoriel avec le graphe d'une représentation faible de cette algèbre.

Nous pouvons donc qualifier les positions relatives des solides de la figure 4 : le parallélépipède supérieur est H-O par rapport au cube, et également H-O par rapport au parallélépipède inférieur. Pour plus de clarté, les cristaux des illustrations sont représentés par des pièces d'un jeu de construction («badaboums»).

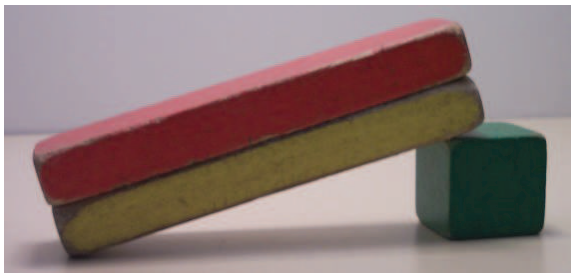


FIG. 4 – Exemple de configuration de cristaux dans l'espace (l'observateur fait face au nord)

Dans le cas de la figure 5, le cube de gauche est H-E par rapport au parallélépipède sur lequel il repose, et le cube de droite est H par rapport au parallélépipède sur lequel il repose. Cependant, dans les deux cas, les objets cubiques reposent horizontalement sur les objets parallélépipédiques. La deuxième algèbre que nous introduisons rend compte de

cette propriété.

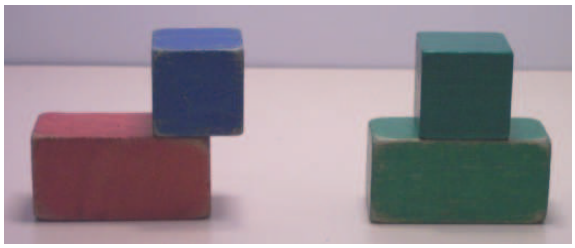


FIG. 5 – Position relative dans l'espace et orientation des cristaux (l'observateur fait face au nord)

Le deuxième algèbre donne à la fois le type de contact entre deux solides (contact direct ou indirect), et la direction de la normale aux plans de contact dans le système précédent lorsque les contacts ont tous la même direction. Si les cristaux représentent des couches géologiques, la direction de cette normale aux plans de contact est le pendage des couches.

Précisons ici ces notions de contact. Les objets géographiques sont physiquement en contact : les couches géologiques sont empilées les unes sur les autres, les bâtiments sont en contact avec le sol, etc. Cependant, lorsque ces objets sont modélisés de manière approchée par des cristaux, ce contact ne se traduit pas nécessairement par le partage géométrique d'une face entre deux cristaux dans notre système. Les cristaux peuvent s'intersecter, et la surface réelle (géographique) de contact est alors associée à deux faces de deux cristaux qui s'intersectent. Le système stocke l'information topologique d'adjacence même si la représentation des cristaux n'a pas une géométrie compatible avec cette adjacence. Notre notion de contact est une notion au sens large, qui inclut les relations topologiques d'adjacence et certains cas d'intersection. Nous définissons alors le plan de contact entre les deux cristaux comme le plan moyen qui sépare les cristaux géométriquement (les faces des cristaux étant planes, il s'agit du plan bissecteur des deux faces cristallines en relation) (Figure 6).

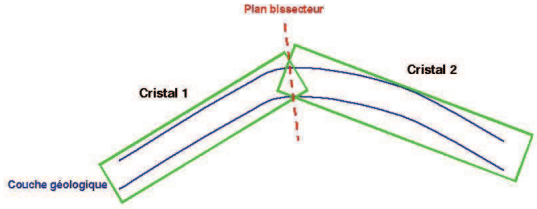


FIG. 6 – Définition du plan de contact de deux cristaux en contact

Dans l'exemple de la figure 5, le cube de gauche est H-E par rapport au parallélépipède sur le lequel il repose, et son contact est H.

2 Relations cardinales dans l'espace

Le problème de l'orientation dans l'espace a déjà été abordé dans la littérature (1; 8). L'approche de Pacheco a été d'étendre le modèle 2D de Zimmerman et Freska en 3D (3). C'est un modèle beaucoup plus fin que celui que nous cherchons à mettre en place, puisqu'en 2D la position relative de deux points définit 15 régions qualitatives.

Il existe également des modèles classiques de relations cardinales, avec des formulations algébriques et des calculs qualitatifs associés (2; 4). Notre approche se situe dans la lignée de ces travaux.

Notre modèle caractérise la position relative de deux cristaux a et b dans l'espace. La troisième dimension étant celle de la verticale dans notre contexte géographique, nous n'utilisons pas le découpage classique en octants, mais un découpage matérialisant la notion d'horizontalité. Nous introduisons donc le vecteur $\vec{ab}$ de $\mathbb{R}^3$ qui relie les centres de symétrie principaux de nos cristaux [1]. Les notions de *haut* et de *bas* (et de *à-niveau*, c'est-à-dire ni en haut ni en bas) sont définies ainsi : si l'angle (en degrés) modulo 180 entre le vecteur $\vec{ab}$ et l'axe $\vec{z}$ est compris entre 0 et 85, b est en haut de a ; si l'angle est compris entre 85 et 95, a et b sont au même niveau ; et si l'angle est compris entre 95 et 180, b est en bas de a. On obtient ainsi une première partition du domaine sur lequel porte notre raisonnement entre trois zones définies par les trois régions d'un cône d'axe $\vec{z}$ et de demi-angle au sommet 85. L'inverse de H est B, et la relation identité est L. Une relation L est toujours complétée d'une direction cardinale classique (NSEO), le cas L seul correspondant à l'identité. Une relation H ou B est complétée d'une direction cardinale classique si le vecteur $\vec{ab}$ est en dehors d'un cone d'axe $\vec{z}$ et de demi-angle 5 degrés. Une relation H, B ou L complétée d'une relation latérale est notée H-?, B-? ou L-? dans cet article.

L'angle du vecteur $\vec{ab}$ avec le nord (que nous assimilerons de manière approchée au vecteur $\vec{y}$ dans les systèmes de coordonnées cartographiques) permet de déterminer la direction cardinale de b par rapport à a (quand le vecteur $\vec{ab}$ est en dehors du cône d'axe $\vec{z}$ et de demi-angle 5 degrés). On divise le plan $(O, \vec{x}, \vec{y})$ en 8 secteurs de 45 degrés, correspondant aux notions intuitives de N, S, E, O, NE, NO, SE et SO. Notons que ce n'est pas l'interprétation la plus naturelle des directions cardinales dans le plan : on considère généralement que le point (x', y') est au nord du point

[1] On pourrait considérer également les barycentres des solides, pour appliquer ce calcul à des solides quelconques, ou lorsque le cristal ne possède pas de centre de symétrie.

(x, y) si $x = x'$ et $y > y'$, et ainsi de suite [2]. Les inverses de ces relations s'obtiennent de manière évidente.

En combinant les relations verticales et latérales, on obtient une partition de l'espace $\mathbb{R}^3$ en 26 zones (Figure 7), et la position du vecteur $\vec{ab}$ donne donc une des 27 relations cardinales possibles (en ajoutant l'égalité L).

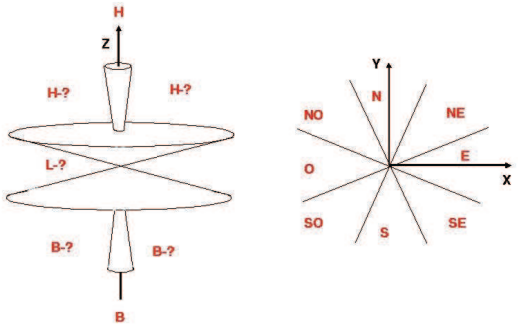


FIG. 7 – Partitionnement de l'espace en directions cardinales 3D, le symbole - ? indique une relation latérale

Pour construire notre formalisme, nous avons utilisé le «kit» de raisonnement qualitatif de Ligozat (5). nous introduisons donc :

- Le domaine D sur lequel on va raisonner. Il s'agit de l'ensemble des positions possibles du centre de symétrie d'un cristal, donc $\mathbb{R}^3$. Les relations binaires R_i sur D sont les sous-ensembles $\{(a, b) \in \mathsf{D}\times\mathsf{D} \mid \vec{ab}$ pointe dans la zone $i\}$. On obtient bien une partition de D×D, avec une relation identité ($a = b$) notée L, et une relation inverse associée à chaque relation.
- Des symboles associés à chacune des relations. Nous avons donc B={ H, H-N, H-S, H-E, H-O, H-NE, H-NO, H-SE, H-SO, L, L-N, L-S, L-E, L-O, L-NE, L-NO, L-SE, L-SO, B, B-N, B-S, B-E, B-O, B-NE, B-NO, B-SE, B-SO }. Nous notons alors A l'ensemble des parties de B.
- Un réseau qui représente la configuration des éléments de D sur lesquels on peut raisonner, avec des arêtes valuées par les symboles. Nous ajoutons également une valuation des arêtes par les vecteurs $\vec{ab}$, de manière à pouvoir faire des inférences quantitatives sur ce graphe, par sommation de vecteurs.
- Une loi de composition de deux relations R_i et R_j, qui donne une nouvelle relation $R_i \circ R_j$. Notons qu'il est également aisé de recalculer une direction cardinale réelle entre deux points. Nous utilisons la table proposée par Frank (2) (Figure 9) pour les relations latérales, et introduisons notre table de composition pour les relations verticales (Figure 8).

	H	B	L	H-?'	B-?'	L-?'
H	H	L	H	H-?'	L-?'	H-?'
B	L	B	B	L-?'	B-?'	B-?'
L	H	B	L	H-?'	B-?'	L-?'
H-?	H-?	L-?	H-?	H-(? ∘ ?')	L-(? ∘ ?')	H-(? ∘ ?')
B-?	L-?	B-?	B-?	L-(? ∘ ?')	B-(? ∘ ?')	B-(? ∘ ?')
L-?	H-?	B-?	L-?	H-(? ∘ ?')	B-(? ∘ ?')	L-(? ∘ ?')

FIG. 8 – Table de composition des relations verticales

Les symboles ? et ?' désignent une des relations cardinales latérales N, S, E, O, NE, NO, SE et SO. Les compositions ? ∘ ?' se déduisent de la table de Frank.

	N	NE	E	SE	S	SW	W	NW	0
N	N	n	ne	o	o	o	nw	nw	N
NE	n	ne	ne	e	o	o	o	n	NE
E	ne	ne	E	e	se	o	o	o	E
SE	o	e	e	SE	se	s	o	o	SE
S	o	o	se	se	S	s	sw	w	S
SW	o	o	o	s	s	SW	sw	w	SW
W	nw	o	o	o	sw	sw	W	w	W
NW	n	n	o	o	o	w	w	NW	NW
0	N	NE	E	SE	S	SW	W	NW	0

FIG. 9 – Table de composition des relations cardinales dans des secteurs coniques de (2) (les lettres minuscules correspondent aux inférences incertaines, le W est pour Ouest et O pour Origine)

En utilisant successivement les deux tables, si b est au H-NE de a et c est au B-E de b, on peut inférer que c est au L-NE de a.

Nous déterminons également à quelle relation le couple (a, c) appartient effectivement en faisant la somme des vecteurs portés par les arêtes du graphe, et en analysant à laquelle des 27 régions de l'espace de vecteur appartient.

En d'autres termes, nous n'arguons pas dans notre cas en faveur d'un seul calcul qualitatif, mais préférons lui associer un calcul incluant des éléments quantitatifs. Notons également que notre définition des directions cardinales (avec un secteur nord plutôt qu'une vraie direction) dessert la fiabilité des inférences, et donc un calcul purement qualitatif.

D'un point de vue formel, l'ensemble de parties d'une ensemble muni des opérations d'intersection, d'union, et de passage au complémentaire, donc A, est une algèbre de Boole (d'un point de vue algébrique, c'est l'aspect treillis booléen, c'est-à-dire borné, distributif et complémenté, qui nous intéresse, car il semble peu utile dans le cas général de faire opérer le corps $\mathbb{Z}/2\mathbb{Z}$ sur l'ensemble A !). Nous transformons cette algèbre en algèbre non-associative comme Ligozat (6), pour construire un calcul qualitatif, et conservons également la possibilité d'un calcul semi-qualitatif,

[2] Notre définition des directions cardinales rend plus ambiguës les compositions, mais respecte mieux à notre avis le caractère qualitatif de ces relations : la probabilité d'observer deux objets pour lesquels $x = x'$ est strictement nulle dans $\mathbb{R}^2$, et très faible dans $\mathbb{Q}^2$ (si on tient compte du fait que les coordonnées sont des nombres rationnels du fait de leur stockage informatique).

c'est-à-dire que nous pouvons effectuer des inférences à partir des compositions de relations, et en recalculant directement à quelle relation chaque composition d'éléments de relations appartiennent (c'est-à-dire en exploitant la représentation de cette algèbre par du calcul vectoriel élémentaire). Cette représentation est la position des centres de symétrie d'un ensemble de cristaux dans $\mathbb{R}^3$. La valuation du graphe par les vecteurs de direction permet de déterminer aisément toutes les compositions. L'algèbre est clairement non-associative. En utilisant le signe conventionnel ; pour la composition de relation, Frank montre que pour les relations latérales (S ; N) ; E = C ; E = E tandis que S ; (N ; E) = S ; (NE) = C.

Le calcul qualitatif est très utile pour détecter les situations impossibles du type B est au nord de A, C est au nord de B et C est au sud de A. Il permet également de faire des inférences fiables dans de nombreux cas. Comme le remarque Frank dans (2), les inférences ne sont correctes à l'aide de sa table que lorsque les distances entre les éléments en relation sont du même ordre de grandeur, ce qui ne sera pas toujours le cas en pratique. Notre approche semi-qualitative permet de lever les ambiguïtés.

3 Liens entre cristaux dans l'espace

Dans de nombreux cas, les cristaux de notre modèle (donc les objets géographiques 3D) sont en contact les uns avec les autres : empilement de couches géologiques, corps de bâtiments adjacents, etc. Dans notre modèle, les contacts entre cristaux sont caractérisés par des plans.

Nous introduisons donc deux notions. Dans le cas de deux cristaux en contact direct, la direction cardinale (dans le système présenté précédemment) de la normale au plan de contact donne l'orientation du contact entre les cristaux. On dispose donc ainsi de deux relations cardinales pour décrire la position relative de ces deux solides : la direction cardinale du deuxième solide par rapport au premier dans l'espace, et la direction cardinale de la normale à la surface de contact. On peut donc discriminer entre les deux cas de superposition de la figure 5.

Dans le cas de deux cristaux en contact indirect, c'est-à-dire par l'intermédiaire d'un ou plusieurs cristaux, on distingue trois cas que l'on qualifie de la manière suivante :

- lien majeur si les cristaux sont en contact direct. On note cette relation AM- ?, ? étant la direction cardinale du contact ;
- lien intermédiaire si les cristaux sont en contact indirect par des plans de contact de même direction ; dans ce cas on donne également la direction cardinale du contact. On note alors cette relation AS- ? ;
- lien mineur si les cristaux sont en contact indirect et le chemin de contact comporte des changements de direction des normales des plans de contact. Nous discutons de la notion de chemin de contact dans la section suivante. Cette dernière relation est notée Am.

En termes algébriques, les relations que nous introduisons sont donc l'adjacence majeure AM ou intermédiaire AS, complétée par la direction cardinale des plans de contact, l'adjacence mineure Am, et l'égalité E des cristaux. On dispose donc de 54 relations pour décrire les contraintes de contact entre deux cristaux en contact direct ou indirect. Notons que les relations AS- ? et Am se déduisent par composition de relations AM, et que l'on ne compose jamais les relations cardinales 3D : on ne fait pas usage de l'agèbre précédentes des relations cardinales dans l'espace. La relation entre deux cristaux en contact n'est AM- ? ou AS- ? que si tous les contacts ont la même direction cardinale.

L'algèbre A des parties de l'ensemble des relations est une algèbre non-associative. L' opération neutre pour la composition est E. Chaque relation a une relation inverse : l'inverse de Am est Am, et les inverses des relations de type AM- ?- ? et AS- ?- ? sont obtenues par inversions des relations cardinales, par exemple l'inverse de AS-H-SE est AS-N-NO. Enfin, nous introduisons l'opération binaire de composition ; définie ainsi :

- Am ∘ Am = Am
- AS- ? ∘ AS- ?' = AS- ? si ?= ?', Am sinon
- AM- ? ∘ AS- ?' = AS- ? si ?= ?', Am sinon
- AM- ? ∘ AM- ?' = AS- ? si ?= ?', Am sinon

Les représentations de cette algèbre se font sur des ensembles de cristaux au minimum deux à deux adjacents. Le graphe de relations est valué par les relations, et les compositions se font à l'aide de l'opération définie précédemment. On s'est donc doté d'un calcul qualitatif. L'algèbre étant associative, il s'agit bien d'une algèbre relationnelle.

4 Définition des chemins de contact

Pour exploiter notre calcul qualitatif, il faut être capable avant tout de caractériser la relation de deux cristaux quelconques en contact direct ou indirect.

Dans le cas de cristaux en contact indirects, ils le sont en général par l'intermédiaire d'un très grand nombre de cristaux. Pour déterminer le chemin de contact qui va nous servir à déterminer la nature de ce contact (c'est-à-dire si on est dans un cas AS- ? ou Am), plusieurs approches sont envisageables. Nous introduisons un graphe dont les nœuds sont les centres de symétrie des cristaux [3], et les arêtes les contacts entre cristaux, valués par la direction cardinale des normales aux plans de contact.

La première approche consiste à explorer l'ensemble des chemins du graphe, et à caractériser le chemin par AS- ? s'il existe un chemin dont les valuations sont identiques. Notons qu'il est facile d'ajouter deux heuristiques :

[3] Là encore on pourrait utiliser les barycentres pour généraliser notre approche.

– dès qu'un lien Am est détecté, on cesse d'explorer ce chemin ;
– dès qu'un chemin AS- ? est identifié, l'algorithme s'arrête.

Il s'agit en fait d'un algorithme similaire à celui de Dijkstra.

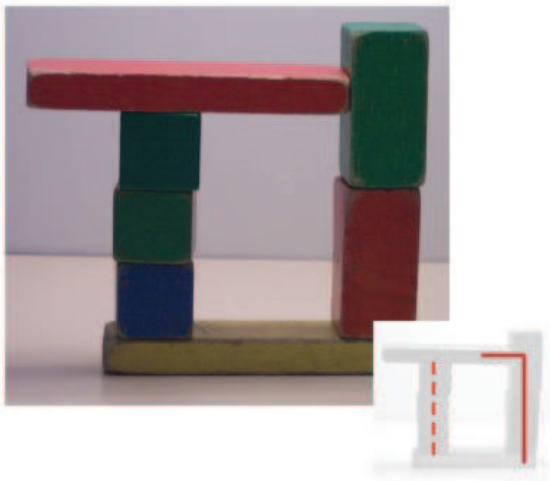


FIG. 10 – Chemins de contact possibles entre deux cristaux

La deuxième approche consiste à utiliser une logique de plus court chemin conventionnelle. La notion de plus court chemin nécessite une valuation correcte des arêtes. Le cas d'école de la figure 10 illustre deux chemins possibles pour calculer la relation entre la barre jaune et la barre rouge du dessus. Le chemin mettant en jeu un nombre minimal de contacts (en trait plein) donne un lien Am entre les cristaux alors qu'il existe un chemin (en trait pointillé) donnant un lien AS-H. On peut valuer les arêtes par la distance euclidienne, pour essayer de réduire ce problème, mais les résultats seront de manière générale moins satisfaisants qu'avec l'approche d'exploration exhaustive du graphe.

5 Exemples de relations cardinales et d'adjacence

Nous illustrons quelques relations cardinales et relations d'adjacence, ainsi que leur composition, avec l'exemple de la figure 11.

Dans cet exemple, l'objet 2 est H-S par rapport à 1. L'objet 3 est L-E par rapport à 2. On en déduit par composition que l'objet 3 est H-SE par rapport à 1. En termes de relations inverses, on a bien sûr que 1 est B-NO par rapport à 3.

En termes de relations d'adjacence, l'objet 3 est en relation directe AM-BE avec 2. L'objet 2 est en relation Am avec l'objet 1, car il n'existe pas de chemin de contact de même direction cardinale (le chemin est H, puis HO). Par composition de AM-BE et Am, l'objet 3 est en relation Am avec l'objet 1.

6 Conclusion

Nous avons présenté dans cet article les premiers travaux sur deux raisonnements qualitatifs et semi-qualitatifs sur la position relative de deux solides dans l'espace. Notre approche se veut simple, et en accord avec la nature de l'information modélisée par ces solides : de l'information géographique de sous-sol, sol ou sur-sol.

Cette approche prend donc en compte le rôle particulier de la troisième dimension en géographie (la verticale, qui est la direction de la force de gravité). Cette contrainte se traduit dans notre modèle de directions cardinales spatiales par le découpage particulier de l'espace en termes de haut et de bas.

La deuxième caractéristique de notre approche est de chercher à qualifier la direction cardinale de la normale au plan de contact entre solides, que ce contact soit direct ou indirect. Cette approche se veut compatible avec une description plus mécanique des contacts, où on introduit des ellipsoïdes de contraintes, et ou les interfaces jouent un rôle particulier dans ces calculs (en mécanique des sols ou des roches par exemple).

Ces deux calculs (semi-qualitatif et qualitatif) permettent de satisfaire des requêtes géographiques, et de donner une idée qualitative de la propagation d'un phénomène. Leur couplage avec un calcul temporel adapté (comme celui d'Allen par exemple) permettra d'aborder le raisonnement qualitatif sur des phénomènes spatio-temporels complexes, à l'aide de notre système (11).

Références

[1] P. Balbiani, J.-F. Condotta et L. Fari nas del Cerro. Spatial reasoning about points in a multidimensional setting. In *Proceedings of IJCAI'99 Spat. & Temp. Reasoning WZ*, pages 105–113, 1999.

[2] A. U. Frank. Qualitative spatial reasoning : Cardinal directions as an example. *Int. J. Geographical Information Systems*, **10**, No. 3, 269–290, 1996.

[3] C. Freksa et K. Zimmermann. On the utilization of spatial structures for cognitively plausible and efficient reasonning. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 1992.

[4] G. Ligozat. Reasoning about cardinal directions. *Journal of Visual Languages and Computing*, **9**, 23–44, 1998.

FIG. 11 – Configuration de cristaux dans l'espace (l'observateur fait face au nord)

[5] G. Ligozat. Categorical methods in qualitative reasoning : the case for weak representations. In *Proceedings of COSIT'05*, volume Springer LNCS 3693, 265-282, 2005.

[6] G. Ligozat et J. Renz. What is a qualitative calculus ? a general framework. In *Proceedings of PRICAI'04*, volume Springer LNCS 3157, 53–64, 2004.

[7] M. Molenaar. A formal data structure for 3d vector maps. In *Proceedings of EGIS'90*, volume 2, pages 770–781, Amsterdam, The Netherlands, 1990.

[8] J. Pacheco, M. T. Escrig et F. Toledo. Qualitative spatial reasoning on three-dimensional orientation point objects. In *Proceedings of the Sixteenth International Workshop on Qualitative Reasoning*, 2002.

[9] S. Pigot. *A topological model for a 3-dimensional Spatial Information System,*. PhD thesis, University of Tasmania, Australia, 1995.

[10] C. Pomerol, Y. Lagabrielle et M. Renard. *Éléments de géologie*. Dunod, 2003.

[11] B. Poupeau et O. Bonin. 3d analysis with high-level primitives : a crystallographic approach. In *Proceedings of SDH'06*, 2006.

[12] S. Zlatanova. *3D GIS for urban development*. PhD thesis, ITC, The Netherlands, 2000.

QAT: une boîte à outils dédiée aux algèbres qualitatives

Jean-François Condotta[1] Gérard Ligozat[2] Mahmoud Saade[1]

[1] CRIL-CNRS, Université d'Artois, 62307 Lens Cedex, France
[2] LIMSI-CNRS, Université d'Orsay, 91403 Orsay, France
ligozat@limsi.fr, {condotta, saade}@cril.univ-artois.fr

Résumé La représentation et le raisonnement sur les informations spatiales et temporelles est une importante tâche dans de nombreuses applications de l'Intelligence Artificielle. Ces vingt dernières années de nombreux formalismes ont été proposés pour la représentation et le raisonnement sur le temps et l'espace à base de contraintes qualitatives. Dans la première partie de ce papier nous proposons et étudions une définition générale de tels formalismes en considérant des relations de base d'arité quelconque. Dans une seconde partie nous décrivons **QAT** (Qualitative Algebra Toolkit), une librairie de programmation par contraintes permettant de gérer des réseaux de contraintes basés sur ces calculs qualitatifs. La principale motivation de ce travail découle du fait que la plupart des logiciels traitant des calculs qualitatifs ont été uniquement implantés pour des calculs qualitatifs spécifiques.

1 Introduction

De nombreux calculs qualitatifs ont été développés ces vingt dernières années pour représenter et raisonner sur les configurations temporelles ou spatiales. La représentation et le raisonnement sur les informations temporelles et spatiales est une tâche importante de nombreuses applications telles que les systèmes d'informations géographiques (GIS), la compréhension du langage naturel, la navigation de robot, la planification temporelle et spatiale. Le raisonnement temporel et spatial qualitatif a pour but de décrire des relations non numériques entre les entités temporelles ou spatiales. Habituellement un calcul qualitatif (1; 19; 14; 18; 11) utilise des éléments particuliers (sous-ensembles d'un espace topologique, points de la droite des rationnels, intervalles de la droite des rationnels,...) pour représenter les entités spatiales ou temporelles du système, et se focalise sur un nombre limité de relations entre ces éléments (telles que des relations topologiques entre les régions ou des relations de précédence entre les points). Chacune de ces relations représente une situation temporelle ou spatiale particulière. Par exemple, considérons le bien connu formalisme temporel d'Allen appelé l'algèbre des intervalles (1). Il utilise des intervalles de la droite des rationnels pour représenter les entités temporelles. Treize relations de base entre ces intervalles sont utilisées pour modéliser les différentes situations qualitatives possibles entre les entités temporelles (voir la figure 1). Par exemple, la relation de base *overlaps* peut être utilisée pour représenter la situation où une première activité débute avant une deuxième activité et se termine pendant que la deuxième se déroule encore. Les treize relations de base sont JEPD (Jointly Exhaustive and Pairwise Disjoint), ce qui signifie que chaque couple d'intervalles satisfait exactement une des treize relations de base.

Les informations temporelles ou spatiales concernant la situation d'un ensemble particulier d'entités peuvent être représentées en utilisant des réseaux de contraintes particuliers appelés réseaux de contraintes qualitatives (Qualitative Constraint Networks en anglais, QCN en abrégé). Chaque contrainte d'un QCN représente un ensemble de configurations qualitatives acceptables entre les entités temporelles ou spatiales et est définie par un ensemble de relations de base. Le problème de la cohérence pour les QCNs consiste à déterminer si un réseau donné possède des solutions satisfaisant ses contraintes ou non. Pour résoudre ce problème, des méthodes basées sur des algorithmes de propagation locale de contraintes ont été définies, en particulier des méthodes basées sur l'algorithme de chemin-cohérence (17; 16).

Tous les calculs qualitatifs existants partagent la même structure, mais, à notre connaissance, les outils logiciels proposés dans la littérature sont spécifiques à des calculs particuliers. QAT (pour Qualitative Algebra Toolkit) a été conçu pour remédier à cette situation. Plus particulièrement, QAT est une librairie de programmation par contraintes JAVA développée au sein du *CRIL-CNRS* à l'université d'Artois. Elle a pour objectif de proposer des outils génériques et ouverts pour définir et manipuler des algèbres qualitatives et des réseaux de contraintes basés sur ces algèbres. Dans ce papier nous donnons et étudions une définition générale d'un formalisme qualitatif d'arité n quelconque. Dans un second temps nous présentons la boîte à outils QAT.

Ce papier est organisé de la manière suivante. Dans la section 2, nous proposons une définition formelle d'un calcul qualitatif. Cette définition est très générale et couvre des formalismes basés sur des relations de base d'arité quelconque. La section 3 est dévolue aux réseaux de contraintes qualitatives. La section 4 est consacrée à la méthode de la $\circ$-

fermeture. Après une introduction de la librairie QAT dans la section 5, nous concluons dans la section 6.

2 Qu'est ce qu'un calcul qualitatif ?

2.1 Les relations

Dans cette section, nous donnons une définition générale de ce qu'est un calcul qualitatif. Un calcul qualitatif d'arité n (avec $n > 1$) considère un ensemble fini $\mathcal{B}$ de k relations d'arité n définies sur un domaine D (généralement infini). Ces relations sont appelées relations de base ou relations atomiques. Les éléments de D sont les valeurs possibles représentant les entités temporelles ou spatiales. Les relations de base de $\mathcal{B}$ correspondent à toutes les configurations possibles entre n entités temporelles ou spatiales. Les relations de $\mathcal{B}$ sont dites JEPD (jointly exhaustive and pairwise distinct), ce qui signifie que tout n-tuple d'éléments de D appartient à exactement une relation de base de $\mathcal{B}$. Plus formellement, nous avons $\mathsf{B}_i \cap \mathsf{B}_j = \emptyset$, pour tout $i, j \in \{1, \ldots, k\}$ tels que $i \neq j$ et $\mathcal{U} = \bigcup_{i \in \{1,\ldots,k\}} \mathsf{B}_i$, avec $\mathcal{U}$ l'ensemble des éléments de D^n. Étant donné un élément de x appartenant à $\mathcal{U}$ et un entier $i \in \{1, \ldots, n\}$, x_i dénotera l'élément de D correspondant au $i^{\text{ème}}$ composant de x. L'ensemble $\mathcal{A}$ est défini comme l'ensemble des relations correspondant à toutes les unions des relations de base. Formellement, $\mathcal{A}$ est défini par $\mathcal{A} = \{\bigcup B : B \subseteq \mathcal{B}\}$. Il est habituel de représenter un élément $B_1 \cup \ldots \cup B_m$ (avec $0 \leq m \leq k$ et $B_i \in \mathcal{B}$ pour chaque i tel que $1 \leq i \leq m$) de $\mathcal{A}$ par l'ensemble $\{B_1, \ldots, B_m\}$ appartenant à $2^{\mathcal{B}}$. De ce fait, nous ne faisons pas de distinction entre $\mathcal{A}$ et $2^{\mathcal{B}}$ dans la suite de ce papier. De plus, nous supposons que pour tout $i, j \in \{1, \ldots, n\}$ tels que $i < j$, il existe une relation dans $\mathcal{A}$, dénotée par Δ_{ij}, telle que $\Delta_{ij} = \{x \in \mathcal{U} : x_i = x_j\}$. Notons que dans la cas binaire Δ_{12} est la relation identité sur D, et elle correspond à une seule relation de base de $\mathcal{B}$.
Comme exemple, considérons le formalisme temporel qualitatif bien connu appelé le calcul d'Allen (1). Ce formalisme utilise les intervalles de la droite des rationnels pour représenter les entités temporelles. Ainsi D est l'ensemble $\{(x^-, x^+) \in \mathbb{Q} \times \mathbb{Q} : x^- < x^+\}$. L'ensemble des relations de base consiste en un ensemble de 13 relations binaires correspondant à toutes les configurations possibles de deux intervalles. Ces relations de base sont illustrées dans la figure 1. Ainsi, nous avons $\mathcal{B} = \{eq, b, bi, m, mi, o, oi, s, si, d, di, f, fi\}$. Chaque relation de base peut être formellement définie en terme de contraintes sur les bornes des intervalles impliqués, par exemple, $m = \{((x^-, x^+), (y^-, y^+)) \in \mathsf{D} \times \mathsf{D} : x^+ = y^-\}$. L'ensemble $\{b, m\} \in 2^{\mathcal{B}}$ correspond à la relation $b \cup m$ de $\mathcal{A}$. De plus, nous avons $\Delta_{12} = \{eq\}$. Comme deuxième exemple, considérons un calcul basé sur des relations de base ternaires appelé l'algèbre des points cycliques

Relation	Symbole	Inverse	Illustration
precedes	b	bi	X Y
meets	m	mi	X Y
overlaps	o	oi	X Y
starts	s	si	X Y
during	d	di	X Y
finishes	f	fi	X Y
equals	eq	eq	X Y

FIG. 1 – Les relations de base du calcul d'Allen.

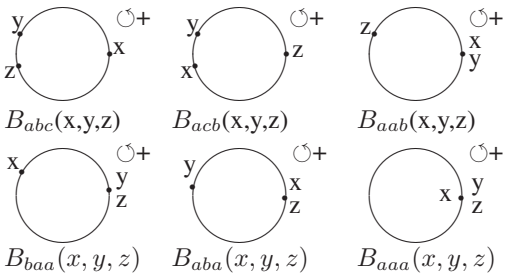


FIG. 2 – Les relations de base de l'algèbre des points cycliques.

(11; 5). Les entités considérées par ce calcul sont les points d'un cercle orienté $\mathcal{C}$. Nous appelons ces points *les points cycliques*. Chaque point cyclique peut être caratérisé par un nombre rationnel de l'intervalle $[0, 360[$. Ce nombre correspond à la mesure de l'arc d'un point origine fixé au point considéré. Ainsi, pour ce calcul, D est l'ensemble des nombres rationnels $\{q \in \mathbb{Q} : 0 \leq q < 360\}$. Dans la suite nous assimilerons un point cyclique au nombre rationnel le représentant. Étant donnés deux points cycliques $x, y \in \mathsf{D}$, $[[x, y]]$ dénotera l'ensemble des valeurs de D correspondant aux points cycliques rencontrés entre x et y quand nous nous déplaçons sur le cercle en suivant son sens d'orientation. Les relations de base de cette algèbre sont les 6 relations ternaires $\{B_{abc}, B_{acb}, B_{aab}, B_{baa}, B_{aba}, B_{aaa}\}$ définies de la manière suivante : $B_{abc} = \{(x, y, z) \in \mathsf{D}^3 : x \neq y, x \neq z, y \neq z$ et $y \in [[x, z]]\}$, $B_{acb} = \{(x, y, z) \in \mathsf{D}^3 : x \neq y, x \neq z, y \neq z$ et $z \in [[x, y]]\}$, $B_{aab} = \{(x, x, y) \in \mathsf{D}^3 : x \neq y\}$, $B_{baa} = \{(y, x, x) \in \mathsf{D}^3 : x \neq y\}$, $B_{aba} = \{(x, y, x) \in \mathsf{D}^3 : x \neq y\}$, $B_{aaa} = \{(x, x, x) \in \mathsf{D}^3\}$. Ces relations sont décrites dans la figure 2. Nous avons $\Delta_{12} = \{B_{aaa}, B_{aab}\}$, $\Delta_{13} = \{B_{aaa}, B_{aba}\}$ et $\Delta_{23} = \{B_{aaa}, B_{baa}\}$.

2.2 Les opérations fondamentales

Comme ensemble de sous-ensembles, $\mathcal{A}$ est muni des opérations ensemblistes classiques parmis lesquelles l'intersection ($\cap$) et l'union ($\cup$). Comme ensemble de relations, il est également muni de l'opération de permutation ($^{\curvearrowleft}$), de l'opération de rotation($^{\frown}$) et de l'opération de composition ($\circ$). Plus particulièrement, étant données deux relations r, s d'arité n, nous définissons l'opération de permutation de la manière suivante :

- $\forall x_1, ..., x_n$, $(x_1, ..., x_n, x_{n-1}) \in r^{\curvearrowleft}$ ssi $(x_1, ..., x_{n-1}, x_n) \in r$.
- $\forall x_1, ..., x_n$, $(x_2, ..., x_n, x_1) \in r^{\frown}$ ssi $(x_1, x_2, ..., x_n) \in r$.

Nous supposons que pour chaque relation de base $B \in \mathcal{B}$, $B^{\curvearrowleft} \in \mathcal{B}$ et $B^{\frown} \in \mathcal{B}$. Étant donnée une relation $R \in 2^{\mathcal{B}}$, nous avons $R^{\curvearrowleft} = \{B^{\curvearrowleft} : B \in R\}$ et $R^{\frown} = \{B^{\frown} : B \in R\}$. Remarquons que dans le cas binaire l'opération de rotation et celle de permutation sont identiques, elles correspondent à l'inverse. Étant données n relations de base $B_1, \ldots, B_n \in \mathcal{B}$, leur *composition qualitative*, dénotée par $\circ(B_1, \ldots, B_n)$ est l'élément de $2^{\mathcal{B}}$ défini de la manière suivante :
Soit $A \in \mathcal{B}$, $A \in \circ(B_1, \ldots, B_n)$ ssi $\exists (x_1, ..., x_n) \in A$ et $\exists u \in \mathsf{D}$ tel que $(x_1, \ldots, x_{n-1}, u) \in B_1$, $(x_1, \ldots, x_{n-2}, u, x_n) \in B_2, \ldots, (u, x_2, \ldots, x_n) \in B_n$. Maintenant si $R_1, \ldots, R_n$ sont n relations dans $2^{\mathcal{B}}$, nous définissons $\circ(R_1, \ldots, R_n)$ par $\{A : A \in \circ(B_1, \ldots, B_n)$ avec $B_1 \in R_1, \ldots, B_n \in R_n\}$. Le calcul des résultats de ces différentes opérations pour les relations de $2^{\mathcal{B}}$ peut être réalisé de manière très efficace en utilisant des tables spécifiant les résultats de ces opérations pour les relations de base de $\mathcal{B}$. Par exemple, considérons les relations $R = \{eq, b, o, si\}$ et $S = \{d, f, s\}$ du calcul d'Allen, nous avons $R^{\frown} = R^{\curvearrowleft} = \{eq, bi, oi, s\}$. La relation $\circ(R, S)$ est $\{d, f, s, b, o, m, eq, si, oi\}$. Considérons maintenant une relation de l'algèbre des points cycliques, soit $R = \{B_{abc}, B_{aab}, B_{aaa}\}$, nous avons $R^{\frown} = \{B_{abc}, B_{aba}, B_{aaa}\}$ et $R^{\curvearrowleft} = \{B_{acb}, B_{aba}, B_{aaa}\}$.

3 Réseaux de contraintes qualitatives

Les réseaux de contraintes qualitatives (QCN en abrégé, pour Qualitative Constraint Networks en anglais) sont utilisés pour exprimer des informations sur les configurations spatiales ou temporelles entre des entités d'un système. Un réseau de contraintes qualitatives consiste en un ensemble de variables et en un ensemble de contraintes sur ces variables. L'ensemble des variables représente les entités temporelles ou spatiales du système. Une contrainte consiste en un ensemble de relations de base acceptables (les configurations possibles) entre des variables. Formellement, un réseau de contraintes qualitatives est défini de la manière suivante :

Définition Un QCN est un couple $\mathcal{N} = (V, C)$ où :

- V est un ensemble fini de m variables où m est un entier positif ;
- C est une application qui associe à chaque n-tuple $(v_1, \ldots, v_n)$ de V^n un sous-ensemble $C(v_1, \ldots, v_n)$ de l'ensemble des relations de base : $C(v_1, \ldots, v_n) \in 2^{\mathcal{B}}$.

$C(v_1, \ldots, v_n)$ est l'ensemble de ces relations de base qui sont permises pour les positions relatives entre les entités représentées par les variables $v_1, \ldots, v_n$. De plus, nous supposons que pour tout $(v_1, \ldots, v_n) \in V^n$ nous avons $C(v_1, \ldots, v_{n-1}, v_n) = C(v_1, \ldots, v_n, v_{n-1})^{\curvearrowleft}$, $C(v_1, \ldots, v_{n-1}, v_n) = C(v_n, v_1, \ldots, v_{n-1})^{\frown}$ et, pour tout $0 < i < j \leq n$, si $v_i = v_j$ alors $C(v_1, \ldots, v_n) \subseteq \Delta_{ij}$. Concernant un QCN $\mathcal{N} = (V, C)$ nous avons les définitions suivantes :

- Une *solution partielle* de $\mathcal{N}$ sur $V' \subseteq V$ est une application σ de V' vers D telle que $(\sigma(v_1), \ldots, \sigma(v_n))$ satisfait $C(v_1, \ldots, v_n)$, pour tout $v_1, \ldots, v_n \in V'$.
- Une *solution* de $\mathcal{N}$ est une solution partielle de $\mathcal{N}$ sur V. $\mathcal{N}$ est *cohérent* si et seulement s'il a une solution.
- Un QCN $\mathcal{N}' = (V, C')$ est un *sous*-QCN de $\mathcal{N}$ (dénoté par $\mathcal{N}' \subseteq \mathcal{N}$) si et seulement si $C'(v_1, \ldots, v_n) \subseteq C(v_1, \ldots, v_n)$ pour tout $v_1, \ldots, v_n \in V$.
- La *restriction* de $\mathcal{N}$ à $V' \subseteq V$ est le QCN $\mathcal{N}' = (V', C')$ avec $C'(v_1, \ldots, v_n) = C(v_1, \ldots, v_n)$ pour tout $v_1, \ldots, v_n \in V'$.
- Un QCN $\mathcal{N}' = (V', C')$ est *équivalent* à $\mathcal{N}$ si et seulement si $V = V'$ et les deux QCN $\mathcal{N}$ et $\mathcal{N}'$ ont les mêmes solutions.
- Le QCN *minimal* de $\mathcal{N}$ est le plus petit (pour $\subseteq$) sous-QCN de $\mathcal{N}$ équivalent à $\mathcal{N}$.
- Un QCN *atomique* est un QCN tel que chaque relation $C(v_1, \ldots, v_n)$ contient une et une seule relation de base.
- Un *scénario cohérent* de $\mathcal{N}$ est un sous-QCN de $\mathcal{N}$ atomique et cohérent.

Étant donné un QCN $\mathcal{N}$, les problèmes principaux considérés sont les suivants :

- décider s'il existe une solution de $\mathcal{N}$;
- trouver une ou plusieurs solutions de $\mathcal{N}$;
- trouver un ou plusieurs scénarios cohérents de $\mathcal{N}$;
- déterminer le QCN minimal de $\mathcal{N}$.

La plupart des algorithmes utilisés pour résoudre ces problèmes sont basés sur une méthode que nous appelons le méthode de $\circ$-fermeture. La section suivante est dévolue à cette méthode.

4 La méthode de ∘-fermeture

La méthode de ∘-fermeture est une méthode de propagation locale de contraintes permettant d'obtenir la propriété de $(0,(n+1))$-cohérence d'un QCN $\mathcal{N} = (V, C)$, ce qui signifie que toutes les restrictions de $\mathcal{N}$ à $(n+1)$-variables sont cohérentes. Remarquons que nous n'obtenons pas forcément la $(n+1)$-cohérence. La méthode de la ∘-fermeture consiste à itérer l'opération suivante : $C(v_1,\ldots,v_n) := C(v_1,\ldots,v_n) \cap \circ(C(v_1,\ldots,v_{n-1},v_{n+1}), C(v_1,\ldots,v_{n-2},v_{n+1},v_n),\ldots, C(v_{n+1},v_2,\ldots,v_n))$, pour toutes (n+1) variables $v_1,\ldots,v_{n+1}$ de V, jusqu'à l'obtention d'un point fixe. Le QCN obtenu est un sous-QCN de $\mathcal{N}$ qui lui est équivalent, et tel que $C(v_1,\ldots,v_n) \subseteq \circ(C(v_1,\ldots,v_{n-1},v_{n+1}), C(v_1,\ldots, v_{n-2},v_{n+1},v_n),\ldots, C(v_{n+1},v_2,\ldots,v_n))$ pour toutes variables $v_1,\ldots,v_{n+1}$ de V. Cette dernière propriété fait que nous dirons que le sous-QCN obtenu est ∘-fermé. Pour des formalismes qualitatifs particuliers cette propriété est équivalente à la propriété de chemin-cohérence (15). Dans le cas où le sous-QCN obtenu par la méthode de la ∘-fermeture contient la relation vide, nous pouvons affirmer que le QCN intial n'est pas cohérent. Cependant, dans le cas contraire, nous ne pouvons pas inférer la cohérence du QCN dans le cas général.

Il y a deux algorithmes bien connus dans la littérature pour obtenir la propriété de chemin-cohérence des CSP discrets (15; 17) : les algorithmes PC1 et PC2. Ces algorithmes ont été adaptés à de nombreuses reprises pour le cas binaire qualitatif afin d'obtenir la ∘-fermeture (2; 21; 13; 8; 10). Une adaptation possible de PC1 au cas n-aire est la fonction $\mathsf{PC1_n}$ définie dans Algorithme 1. La fonction $\mathsf{PC2_n}$ définie dans Algorithme 2 est inspirée de PC2. La complexité en temps de $\mathsf{PC1_n}$ est $O(m^{2n+1})$, tandis que celle de $\mathsf{PC2_n}$ est $O(m^{n+1})$. Malgré cela, $\mathsf{PC2_n}$ peut être en pratique moins efficace que $\mathsf{PC1_n}$. Cela est principalement dû au fait que $\mathsf{PC2_n}$ effectue une initialisation coûteuse de la queue Q (ligne 2). Cette étape peut prendre plus de temps que le traitement des éléments dans la queue, en particulier pour les QCN non cohérents. C'est pourquoi nous avons introduit la fonction PCMixed (voir Algorithme 3) pour remédier à cet inconvénient. PCMixed réalise une première étape correspondant à une boucle de $\mathsf{PC1_n}$ puis continue de la manière de $\mathsf{PC2_n}$. Pour clore cette section, remarquons que les fonctions $\mathsf{PC2_n}$ et PCMixed peuvent être améliorées de manière conséquente en utilisant des heuristiques pour la sélection du chemin à traiter (ligne 3 et ligne 10). Par exemple, nous pouvons d'abord séléctionner en premier les chemins contenant les contraintes avec les plus petites cardinalités (voir (9)).

Algorithme 1

Function $\mathsf{PC1_n}(\mathcal{N})$, avec $\mathcal{N} = (\{v_1,\ldots,v_m\}, C)$.

1: **repeat**
2: $change \leftarrow$ **false**
3: **for** $j \leftarrow 1$ to m **do**
4: **for** $i_1 \leftarrow 1$ to m **do**
5: ...
6: **for** $i_n \leftarrow 1$ to m **do**
7: **if** $revise(i_1,...,i_n,j)$ **then**
8: **if** $C(v_{i_1},...,v_{i_n}) = \emptyset$ **then return false**
9: **else** $change \leftarrow$ **true**
10: **until not** $change$
11: **return true**

Function *revise*$(i_1,...,i_n,j)$.

1: $R \leftarrow C(v_{i_1},\ldots,v_{i_n}) \cap \circ(C(v_{i_1},\ldots,v_{i_{n-1}},v_j),$
2: $C(v_{i_1},\ldots,v_{i_{n-2}},v_j,i_n),\ldots,C(v_j,v_{i_2},\ldots,v_{i_n}))$
3: **if** $C(v_{i_1},...,v_{i_n}) \subseteq R$ **then return false**
4: $C(v_{i_1},...,v_{i_n}) \leftarrow R$
5: updateRelations($C(v_{i_1},...,v_{i_n})$)
6: **return true**

Algorithme 2

Function $\mathsf{PC2_n}(\mathcal{N})$, avec $\mathcal{N} = (\{v_1,\ldots,v_m\}, C)$.

1: $Q \leftarrow \bigcup_{0<i_1,\ldots,i_n\leq m} relatedPaths(i_1,...,i_n)$
2: **while** $Q \neq \emptyset$ **do**
3: select and delete a path $(i_1,...,i_{n+1})$ from Q
4: **if** $revise(i_1,...,i_{n+1})$ **then**
5: **if** $C(v_{i_1},...,v_{i_n}) = \emptyset$ **then return false**
6: **else** $Q \leftarrow Q \bigcup relatedPaths(i_1,...,i_n)$
7: **return true**

Function relatedPaths$(i_1,...,i_n)$.

1: $Q \leftarrow \emptyset$
2: **for** $j \leftarrow 1$ to n **do**
3: **for** $k \leftarrow 1$ to m **do**
4: $Q \leftarrow Q \bigcup \{(i_1,...,i_{j-1},k,i_{j+1},...,i_n,i_j)\}$
5: **return** Q

5 La boîte à outils pour les algèbres qualitatives QAT

Clairement, tous les calculs qualitatifs existants partagent la même structure. À notre connaissance, les différentes implantations et outils logiciels proposés dans la littérature sont dédiés à des calculs particuliers. QAT (Qualitative Algebra Toolkit) a été développé pour remédier à cette situation. Plus particulièrement, QAT est une librairie de programmation par contraintes écrite en JAVA et développée au *CRIL-CNRS* à l'université d'Artois. L'objectif de cette librairie est de proposer des outils génériques et ouverts pour définir et manipuler des algèbres qualitatives et des réseaux de contraintes basés sur ces algèbres. Le noyau de QAT contient trois principaux paquetages. Dans le reste de cette section nous présentons chacun de ces paquetages.

Algorithme 3

Function PCMixed($\mathcal{N}$), avec $\mathcal{N} = (\{v_1, \ldots, v_m\}, C)$.

1: $Q \leftarrow \emptyset$
2: **for** $j \leftarrow 1$ to m **do**
3: **for** $i_1 \leftarrow 1$ to m **do**
4: ...
5: **for** $i_n \leftarrow 1$ to m **do**
6: **if** $revise(i_1, ..., i_n, j)$ **then**
7: **if** $C(v_{i_1}, ..., v_{i_n}, v_j) = \emptyset$ **then return false**
8: **else** $Q \leftarrow Q \bigcup relatedPaths(i_1, ..., i_n)$
9: **while** $Q \neq \emptyset$ **do**
10: select and delete a path $(i_1, ..., i_n, j)$ from Q
11: **if** $revise(i_1, ..., i_n, j)$ **then**
12: **if** $C(v_{i_1}, ..., v_{i_n}) = \emptyset$ **then return false**
13: **else** $Q \leftarrow Q \bigcup relatedPaths(i_1, ..., i_n)$
14: **return true**

5.1 Le paquetage Algebra

Le premier paquetage traite des aspects algébriques des calculs qualitatifs. Tandis que les programmes proposés dans la littérature pour utiliser les formalismes qualitatifs sont des implantations *ad hoc* pour des algèbres qualitatives spécifiques et des méthodes de résolution spécifiques, QAT permet à l'utilisateur de définir des algèbres qualitatives quelconques (algèbres non-binaires incluses) en utilisant un simple fichier XML. Ce fichier XML, qui respecte une DTD spécifique, contient les définitions des différents éléments formant la structure algébrique du calcul qualitatif : l'ensemble des relations de base, les éléments diagonaux, la table de rotation, la table de permutation et la table de composition. Nous avons défini ce fichier XML pour la plupart des algèbres qualitatives de la littérature : l'algèbre des intervalles (1), l'algèbre des points (21), l'algèbre des points cycliques (5), l'algèbre des intervalles cycliques (4), l'algèbre des rectangles (6), l'algèbre INDU (18), l'algèbre des points multidimensionnels (7), l'algèbre RCC-5 (19), l'algèbre RCC-8 (19), l'algèbre des relations cardinales (14). Des outils permettant de définir une algèbre qualitative comme le produit cartésien d'algèbres sont aussi disponibles. La paquetage contient aussi une classe permettant de définir et de manipuler des relations des algèbres qualitatives. Puisque la plupart des algèbres qualitatives utilisées dans la littérature sont basées sur des relations d'arité 2, nous avons particularisé cette classe en une classe permettant un traitement spécifique des relations d'arité 2. Plus généralement, un certain nombre de classes génériques de QAT ont été spécialisées pour les relations binaires afin de proposer des méthodes plus efficaces pour les calculs d'arité 2.

5.2 Le paquetage QCN

Ce paquetage contient des outils pour définir et manipuler des réseaux de contraintes qualitatives définis sur des algèbres qualitatives quelconques. Comme pour la structure algébrique d'une algèbre, une DTD permet de définir des fichiers XML specifiant des QCN. Le fichier XML liste les variables et les relations définissant les contraintes qualitatives. Des fonctionnalités sont proposées pour manipuler les variables, les contraintes et les relations de base qu'elles contiennent. Par exemple, nous définissons des classes correspondant à des itérateurs pour accéder aux contraintes d'un QCN, ou pour accéder aux relations de base d'une contrainte satisfaisant certains critères. Une partie du paquetage QCN est dévolue à la génération d'instances aléatoires de QCNs. Une grande part des recherches dans le domaine des calculs qualitatifs consiste en l'élaboration de nouveaux algorithmes pour résoudre des QCN. L'efficacité de ces algorithmes doit être validée à l'aide d'expérimentations sur des instances de QCN. Malheureusement, dans la cas général il n'existe pas d'instances issues de problèmes du monde réel. Ainsi, la génération d'instances aléatoires est une tâche nécessaire (9). Le paquetage QCN de QAT fournit des modèles génériques permettant de générer des instances aléatoires de QCN pour des calculs qualitatifs quelconques.

Algebra	algèbres qualitatives relations ...
QCN	réseaux de contraintes contraintes itérateurs de réseaux itérateurs de contraintes génération de QCN ...
Solver	méthodes pour la cohérence méthodes de recherche de solutions méthodes pour la minimalité méthodes de propagation heuristiques ...

FIG. 3 – Les trois principaux paquetages de QAT.

5.3 Le paquetage Solver

Ce paquetage contient de nombreuses méthodes pour résoudre les principaux problèmes concernant les réseaux de contraintes qualitatives : le problème de la cohérence, le problème de trouver une ou plusieurs solutions, et le problème de la minimalité. Toutes ces méthodes sont géné-

riques et peuvent être appliquées sur des QCN basés sur des calculs qualitatifs quelconques. Elles utilisent l'aspect algébrique des calculs sans considérer la sémantique des relations de base. En d'autres termes, elles font abstraction des définitions des relations de base et manipulent seulement *les symboles* correspondant aux relations. Néanmoins, en utilisant le modèle orienté objet, il est très facile de particulariser une méthode de résolution pour une algèbre qualitative spécifique ou un type particulier de relations. Nous avons implanté la plupart des méthodes de résolution classiques, telles que les méthodes de type *generate and test*, les méthodes de recherches basées sur le *backtrack and forward checking*, et les méthodes de propagation locale de contraintes. L'utilisateur peut configurer ces différentes méthodes en sélectionnant les différentes heuristiques à utiliser. Ces heuristiques diffèrent sur la manière de choisir les variables ou les contraintes à parcourir, et celle de sélectionnner les relations de base d'une contrainte durant la recherche. L'ordre dans lequel les contraintes sont sélectionnées et l'ordre dans lequel les relations de base sont examinées peut grandement affecter les performances d'un algorithme de *backtrack* (9). Le principe des heuristiques d'ordonnancement de contraintes est de sélectionner les contraintes les plus restrictives d'abord. L'idée derrière l'odronnancement des relations de base est d'ordonner les contraintes de manière à ce que les valeurs les plus susceptibles de conduire à une solution soient tout d'abord sélectionnées. QAT permet à l'utilisateur d'implanter facilement de nouvelles heuristiques basées sur des heuristiques existantes. Concernant les méthodes de propagation locale de contraintes, tandis que dans le cas des CSP discrets l'arc cohérence est très largement utilisée (3), la méthode de la $\circ$-fermeture est la plus efficace et la plus fréquemment utilisée des méthodes de propagation locales de contraintes dans le domaine des contraintes qualitatives. Plus exactement, les méthodes utilisées sont basées sur une propagation locale de contraintes utilisant la composition qualitative, à la manière des algorithmes $\mathsf{PC1_n}$ et $\mathsf{PC2_n}$ décrits dans la section précédente.

5.4 Paquetages supplémentaires

En plus de ces trois principaux paquetages, QAT en contient d'autres moins fondamentaux et plus applicatifs. Nous pouvons mentionner la paquetage Campaign permettant d'implanter des outils pour réaliser des tests pour évaluer de nouvelles méthodes de résolution ou des nouvelles heuristiques. Comme autre exemple nous pouvons également mentionner le paquetage Merging qui contient des classes permettant la fusion d'informations temporelles ou spatiales représentées par plusieurs QCNs, similaires aux opérations de fusion utilisées pour les bases de connaissance propositionnelles (20; 12).

6 Conclusion

Dans ce papier nous avons proposé et étudié une définition formelle et générale des calculs qualitatifs basés sur des relations de base d'arité quelconque. Cette définition unificatrice nous permet de capturer la structure algébrique de tous les calculs qualitatifs de la littérature. Les éléments principaux de cette structure algébrique sont les éléments diagonaux et les opérations de permutation, de rotation et de composition qualitative. Dans une seconde partie nous décrivons QAT (Qualitative Algebra Toolkit), une librairie de programmation par contraintes JAVA permettant la gestion de réseaux de contraintes définis sur des calculs qualitatifs d'arité n quelconque. Cette boîte à outils fournit des méthodes pour résoudre le problème de la cohérence et des problèmes liés, aussi bien que des heuristiques utilisées dans le domaine. Comme QAT est implantée dans un langage utilisant la technologie orienté objet, c'est une plateforme ouverte et ses fonctionnalités peuvent être facilement étendues. De nouvelles heuristiques (resp. méthodes) peuvent être définies et testées. Parmi les outils proposés se trouvent également des classes permettant de générer et d'utiliser des bancs d'essai pour les réseaux de contraintes qualitatives. Ainsi de nouvelles heuristiques ou de nouveaux algorithmes de résolution peuvent être facilement évalués. La documentation et le source de la librairie QAT peuvent être trouvés à `http://www.cril.univ-artois.fr/~saade/QAT`.

Références

[1] J. F. Allen. An interval-based representation of temporal knowledge. In *Proc. of the Seventh Int. Joint Conf. on Artificial Intelligence (IJCAI'81)*, pages 221–226, 1981.

[2] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11) :832–843, 1983.

[3] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[4] P. Balbiani et A. Osmani. A model for reasoning about topologic relations between cyclic intervals. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 2000.

[5] P. Balbiani, J.-F. Condotta, et G. Ligozat. Reasoning about cyclic space : Axiomatic and computational aspects. In *Proceedings of Spatial Cognition 2003, LNCS 2685*, pages 348–371, 2003.

[6] P. Balbiani, J.-F. Condotta, et L. Fariñas del Cerro. A new tractable subclass of the rectangle algebra. In

T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 442–447, 1999.

[7] P. Balbiani, J.-F. Condotta, et L. Fariñas del Cerro. Spatial reasoning about points in a multidimensional setting. In *Proceedings of the workshop on temporal and spatial reasoning (IJCAI'99)*, pages 105–113, 1999.

[8] P. van Beek. Reasoning About Qualitative Temporal Information. *Artificial Intelligence*, 58(1-3) :297–326, 1992.

[9] P. van Beek et D. W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4 :1–18, 1996.

[10] C. Bessière. A Simple Way to Improve Path Consistency Processing in Interval Algebra Networks. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, volume 1, pages 375–380, 1996.

[11] A. Isli et A. G. Cohn. A new approach to cyclic ordering of 2D orientations using ternary relation algebras. *Artificial Intelligence*, 122(1–2) :137–187, 2000.

[12] S. Konieczny et R. Pino Pérez. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento*, pages 488–498, 1998.

[13] P. B. Ladkin et A. Reinefeld. Effective solution of qualitative interval constraint problems. *Artificial Intelligence*, 57(1) :105–124, 1992.

[14] G. Ligozat. Reasoning about cardinal directions. *Journal of Visual Languages and Computing*, 1(9) :23–44, 1998.

[15] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence, 1977*, 8 :99–118, 1977.

[16] A. K. Mackworth et E. C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problem. *Artificial Intelligence*, 25(1) :65–74, 1985.

[17] U. Montanari. Networks of constraints : Fundamental properties and application to picture processing. *Information Sciences*, 7(2) :95–132, 1974.

[18] A. K. Pujari, G. Vijaya Kumari, et A. Sattar. INDU : An interval and duration network. In *Australian Joint Conference on Artificial Intelligence*, pages 291–303, 1999.

[19] D. A. Randell, Z. Cui, et A. G. Cohn. A spatial logic based on regions and connection. In *Proc. of the 3rd Conf. on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176, 1992.

[20] P. Z. Revesz. On the semantics of theory change : arbitration between old and new information. In 12^{th} *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databases*, pages 71–92, 1993.

[21] M. Vilain et H. Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *Proc. of the Fifth Nat. Conf. on Art. Int. (AAAI'86)*, pages 377–382, 1986.

Un modèle d'adjacence conceptuelle pour la recherche d'information géographique

Franck Dumoncel

Département informatique,
Laboratoire GREYC,
Équipe Données Document et Langue,
Bâtiment Sciences 3, Boulevard Maréchal Juin 14000 Caen
http://www.greyc.unicaen.fr/
franck.dumoncel@info.unicaen.fr

Résumé Au sein du projet Antéserveur Géographique nous développons une interface intuitive pour spécifier des requêtes spatiales : l'utilisateur y exprime sa requête à l'aide d'un schéma. Après avoir décrit les problèmes d'ambiguïté que pose l'interprétation du schéma, nous proposons une méthode d'interaction avec l'utilisateur pour lever ces ambiguïtés. Pour mettre en œuvre cette interaction nous présentons un algorithme qui calcule des schémas ressemblant au schéma dessiné par l'utilisateur, permettant ainsi de proposer à l'utilisateur des glissements de sens de l'interprétation de son schémas vers d'autres interprétations. Enfin nous présentons notre réalisation informatique ainsi que l'expérimentation qu'elle va permettre d'effectuer.

Mots-clés : Analyse spatiale qualitative, modèle 9-intersections, graphe d'adjacence conceptuelle, interface système-utilisateur,

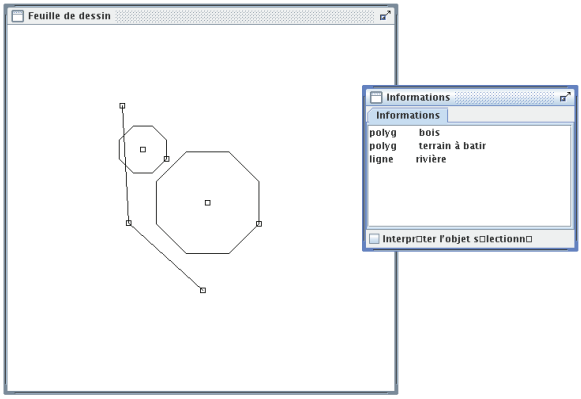


FIG. 1 – Réalisation d'un schéma pour exprimer une requête

1 Introduction

L'information géographique électronique se trouve sous des formes variées : pages HTML, entrepôts de données, bases de données, etc. Les interfaces pour y rechercher de l'information sont soit peu précises soit complexes pour un utilisateur non expert en informatique. Le projet Antéserveur Géographique (2; 1) vise à construire un système de recherche d'information géographique masquant l'hétérogénéité des sources d'information et proposant une interface à la fois simple et précise. Cette interface propose à l'utilisateur de spécifier sa requête à l'aide d'un schéma. Il dispose de trois formes géométriques, polygone, ligne et point, pour dessiner. Sur le schéma, chaque forme géométrique représente une entité géographique spécifiée par l'utilisateur. La disposition de ces entités géographiques les unes par rapport aux autres permet de spécifier des contraintes spatiales.

Prenons l'exemple d'un utilisateur cherchant un terrain à bâtir proche d'un bois, traversé par une rivière. Il exprime alors sa requête comme sur la figure 1. Le petit octogone représente le terrain, un autre, plus grand, représente un bois. L'utilisateur les place proches l'un de l'autre pour spécifier la proximité. Une ligne coupant le terrain représente la rivière. Le système doit alors interpréter ce schéma conformément à ce que l'utilisateur y a exprimé. Nous nous trouvons alors face à deux ambiguïtés : sur-spécification et spécification incertaine du schéma. Sur l'exemple, le terrain est-il en dehors du bois, proche du bois, au nord-ouest du bois ? Faut-il tenir compte de la relation entre la rivière et le bois ? La rivière traverse le terrain par le bord, faut-il en tenir compte ? Toutes ces questions sur l'interprétation du schéma sont des ambiguïtés de sur-spécification.

Considérons maintenant que notre système soit capable de relever quatre concepts différents entre deux zones proches : *touche*, *touche presque*, *proche* et *dans la périphérie*, ces 4 termes correspondant à 4 configurations spatiales ressemblantes mais néanmoins distinctes. L'utilisateur veut exprimer le terme *proche* mais son schéma peut correspondre à une configuration interprétée différemment par le système. Il s'agit là d'une ambiguïté due à la spécification incertaine du schéma.

Pour lever ces ambiguïtés, nous allons utiliser la complémentarité du texte sur le schéma. Le système va retourner à l'utilisateur une interprétation du schéma faite de courtes phrases appelées *microtextes* (7). Chaque microtexte décrit une relation spatiale entre deux entités géographiques. Sur notre exemple, le système peut retourner les trois microtextes suivants :

La rivière ne passe pas par le bois.
La rivière passe par le terrain.
Le terrain est disjoint du bois.

À la lecture de ces microtextes, l'utilisateur s'aperçoit qu'aucun d'entre eux ne lui convient. Nous allons donc mettre en place un processus interactif dans lequel nous allons proposer à l'utilisateur d'autres microtextes correspondant à d'autres interprétations. L'utilisateur pourra ainsi choisir parmi différentes interprétations, faisant ce que nous appellerons un *glissement de sens*. Grâce à cette interaction, les ambiguïtés de sur-spécification et de spécification incertaine doivent être levées.

Le premier microtexte stipule que la rivière ne passe pas par le bois. L'utilisateur ne souhaite rien spécifier entre ces deux entités. Il faut alors être en mesure de lui proposer le glissement : *La rivière ne passe pas par le bois→peu importe la relation entre la rivière et le bois*. Le second microtexte dit que la rivière passe par le terrain, l'utilisateur ne veut pas que son terrain soit coupé en deux et a dessiné pour cela une rivière passant par le bord du terrain, il voudrait donc qu'il en soit tenu compte. Il faut donc pouvoir lui proposer le glissement : *La rivière passe par le terrain→La rivière passe par le bord terrain*. Enfin le troisième microtexte ne tient compte que de la topologie entre le terrain et le bois, il faudrait qu'il soit tenu compte de leur proximité. Entre deux zones proches, nous avons vu que le système est capable de relever quatre concepts représentés figure 2. Le schéma que l'utilisateur a dessiné correspond au concept *touche presque*, le glissement de sens résultant serait alors : *Le terrain est disjoint du bois→Le terrain touche presque le bois* et il ne satisferait toujours pas l'utilisateur. Nous sommes ici face à un problème de spécification incertaine qui peut être résolu en proposant une interprétation ne portant pas sur son schéma mais sur un schéma conceptuellement proche. Cette notion de proximité conceptuelle entre schémas est représentée par un graphe d'adjacence conceptuelle comme celui de la figure 2. Par déplacement dans le graphe nous sommes alors en mesure de considérer un schéma voisin de celui dessiné et ainsi de proposer le glissement : *Le terrain est disjoint du bois→Le terrain est proche du bois*

Grâce à ces trois glissements, l'utilisateur est alors en mesure de transformer les micotextes initiaux pour obtenir ces trois nouveaux microtextes :

Peu importe la relation entre la rivière et le bois.
La rivière passe par le bord du terrain.
Le terrain est proche du bois.

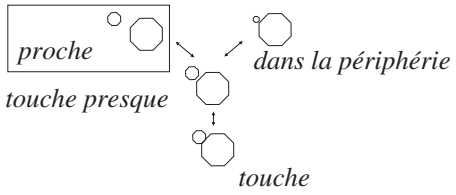


FIG. 2 – Des schémas conceptuellement proches du schéma dessiné servent à proposer des alternatives au schéma de l'utilisateur.

Cette fois-ci il y a bien convergence entre ce que cherche l'utilisateur et ce que le système à interprété.

Pour mettre en place une telle interaction, il est nécessaire de disposer d'un modèle d'analyse spatiale capable de relever différents aspects géographiques (topologie, distance) et aussi capable de retourner une interprétation non seulement du schéma dessiné mais aussi d'un schéma conceptuellement proche. C'est ce que nous verrons dans les deux sections suivantes où nous présenterons notre modèle d'analyse spatiale basé sur le modèle 9-intersection de Max J. Egenhofer. Nous présenterons alors la réalisation informatique qui implante ce modèle et qui se présente, grâce à sa modularité, comme un outil d'étude des concepts spatiaux. Nous présenterons enfin un projet d'expérimentation utilisant cet outil.

2 L'analyse du schéma

Nous allons ici proposer une méthode utilisant le modèle 9-intersections de Max J. Egenhofer de façon récurrente permettant ainsi de relever deux aspects géographiques : topologique et métrique. Nous présenterons d'abord le modèle 9-intersections, ensuite nous verrons l'utilisation récurrente qui en est faite.

2.1 le modèle 9-intersections

Le modèle 9-intersections (3; 4; 6) s'applique sur trois types d'objets : zone, ligne et point. Il décompose chacun de ces objets en trois parties : intérieur, frontière extérieur. De manière informelle ces parties sont définies comme suit :

- *Intérieur :*
  - *Zone* : toute la partie délimitée par le contour de la zone
  - *Ligne* : le tracé de la ligne
  - *Point* : le point lui même
- *Frontière :*
  - *Zone* : le contour de la zone
  - *Ligne* : ses deux extrémités
  - *Point* : le point lui même (la frontière est confondue avec l'intérieur)
- *Extérieur :*

– *Zone* : le reste du plan en dehors du contour
– *Ligne* : le reste du plan en dehors de la ligne
– *Point* : le reste du plan en dehors du point

La relation spatiales entre deux objets est alors représentée par une matrice d'intersections entre les trois parties de chacun des deux objets. Chaque case de la matrice représente une partie d'un objet avec une autre partie de l'autre objet. Elle prend la valeur 1 s'il y a intersection entre ces deux parties, 0 sinon. Deux relations entre deux objets sont représentées figure 3, pour plus de lisibilité les 1 sont représentés par une case noire, les 0 par une case blanche.

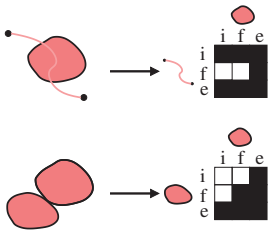


FIG. 3 – Le modèle 9-intersections de Max J.Egenhofer.

En procédant ainsi, ce modèle relève huit configurations entre deux zones (voir figure 4) et 19 configurations entre une zone et une ligne (voir figure 5). Ce modèle est strictement topologique et ne tient absolument pas compte des distances entre les objets.

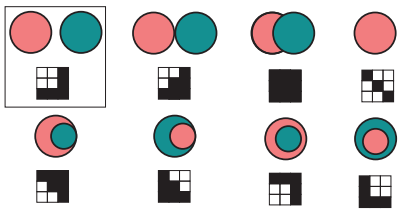


FIG. 4 – Les huit configurations entre deux zones.

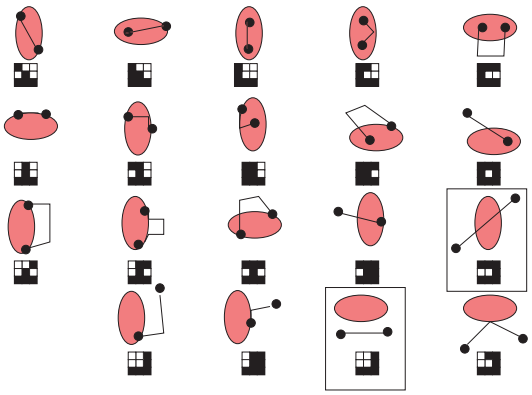


FIG. 5 – Les 19 configurations entre une zone et une ligne.

En utilisant ce modèle pour notre interface, les configurations du schéma correspondent à celle encadrées sur les figures 4 et 5, nous sommes alors en mesure de retourner à l'utilisateur notre première interprétation :

> *La rivière ne passe pas par le bois.*
> *La rivière passe par le terrain.*
> *Le terrain est disjoint du bois.*

Aucun de ces microtextes ne lui convient, il faut alors lui permettre d'effectuer des glissements de sens. Le premier glissement de sens : *La rivière ne passe pas par le bois→peu importe la relation entre le rivière et le bois* peut être proposé sans difficulté, il consiste à ne plus faire d'analyse spatiale entre ces deux entités. Le second glissement : *La rivière passe par le terrain→La rivière passe par le bord du terrain* nécessite de prendre en compte la position de la rivière sur le bord du terrain. Le modèle 9-intersections que nous avons présenté considère l'intérieur de la zone comme un tout sans dissocier le bord du centre. Il n'est donc pas capable de faire cette distinction. Pour remédier à ce problème nous proposons une utilisation récurrente du modèle basée sur la composition et la décomposition des objets.

2.2 Les objets complexes

Dans le modèle 9-intersections les zones ont un intérieur, une frontière et un extérieur. Pour pouvoir relever des distances par rapport à une zone nous allons leur ajouter un centre et une périphérie. Nous obtenons alors des zones dites en «oignons» comme sur la figure 6. Une telle zone sera nommée *objet complexe* par opposition aux objets à trois parties qui seront alors nommés *objets simples*. Un objet complexe est en fait une composition d'objets simples : le centre ou la périphérie, pris isolément sont des objets simples avec un intérieur (l'intérieur du centre, l'intérieur de la périphérie) et un extérieur (l'extérieur du centre, l'extérieur de la périphérie).

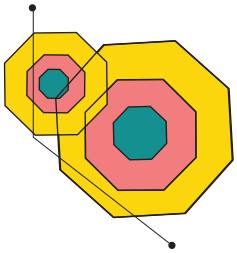


FIG. 6 – La composition des zones en oignons.

Nous venons de voir une première étape qui consiste à faire des compositions d'objets simples pour obtenir des objets complexes. En ce qui concerne l'analyse spatiale d'un objet complexe avec un objet simple ou de deux objets complexes entre eux, les objets complexes seront décomposés,

nous retrouverons ainsi des relations entre objets simples qui pourront être interprétées séparément.

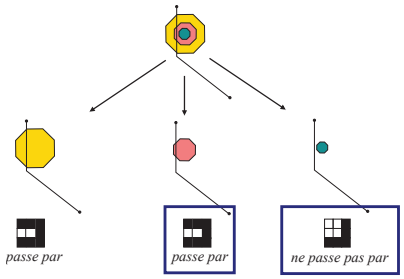


FIG. 7 – La décomposition de la relation entre la rivière et le terrain.

Prenons le relation entre la rivière et le terrain qui est une zone complexe, figure 7. Avec une telle décomposition, nous pouvons observer sur la figure 6 que la rivière passe par le terrain mais pas par son centre : on peut alors en déduire qu'elle passe par son bord. De même, en décomposant les deux objets complexes représentant le bois et le terrain (figure 8), nous observons que la périphérie du terrain coupe le bois et que la périphérie du bois coupe le terrain. Une telle configuration relève une proximité importante entre les deux zones, d'où le terme *touche presque* qui peut lui être associé.

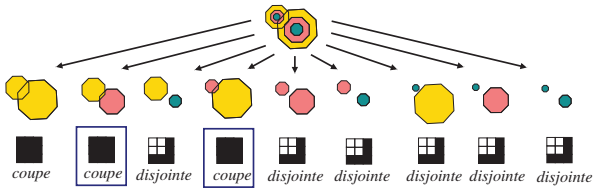


FIG. 8 – La décomposition de la relation entre le terrain et le bois.

Avec une telle analyse sur les objets complexes nous sommes alors en mesure de proposer les glissements de sens permettant de tenir compte des distances exprimées entre la rivière, le terrain et le bois :

> *Peu importe la relation entre la rivière et le bois.*
> *La rivière passe par le bord du terrain.*
> *Le terrain touche presque le bois.*

Comme nous l'avons vu, la contrainte sur la distance entre le terrain et le bois ne satisfait pas l'utilisateur, il voulait spécifier que le terrain était proche du bois. Il faut alors pouvoir lui proposer d'autres glissements de sens correspondant à des configurations conceptuellement proches de la configuration du schéma. C'est ce problème que nous allons aborder dans la section suivante.

3 L'adjacence conceptuelle

Pour remédier à des divergences d'interprétation d'une configuration entre l'utilisateur et le système, nous avons vu qu'il était nécessaire de proposer non seulement une interprétation du schéma de l'utilisateur mais aussi des interprétations de schémas conceptuellement proches de celui-ci. C'est de cette «proximité conceptuelle» que nous allons traiter dans cette section.

3.1 Le principe du mouvement continu

Le principe que nous allons retenir pour déterminer nos adjacences est celui du mouvement continu : deux configurations seront liées dans un graphe d'adjacence conceptuelle si on peut passer de l'une à l'autre par un mouvement continu sans passer par une configuration intermédiaire (5; 8). Prenons l'exemple de la figure 9, deux zones disjointes se rapprochent l'une de l'autre. À un instant, leurs frontières se touchent, une arête est alors posée entre ces deux configurations. Sur la figure ces arêtes sont représentées par des flèches en trait continu. En continuant le mouvement, les deux zones se recouvrent partiellement, une arête est à nouveau ajoutée entre cette nouvelle configuration et la précédente et ainsi de suite. De tels glissements seront appelés *glissements simples*.

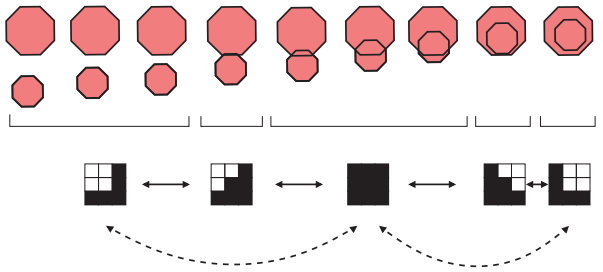


FIG. 9 – Le principe du glissement continu entre deux objets. Les flèches en trait plein représentent les glissements simples, les flèche en trait interrompu représentent les sauts.

Les glissements simples tiennent compte des frontières. En ce qui concerne le centre et la périphérie d'une zone, nous ne voulons pas prendre en compte leurs frontières, pour ces parties, seul compte d'être dedans ou dehors : la périphérie d'une ville ou le centre d'un bois n'ont pas de frontières précises. Aux glissements simples nous ajouterons donc des *sauts* : toujours basés sur le principe du mouvement continu, les sauts ne prendront plus en compte les configurations où seule la relation avec la frontière change. Sur la figure 9, les sauts sont représentés par des flèches en traits interrompus. En appliquant le principe des glissements simples et des sauts sur les huit configurations zone-zone nous obtenons le graphe de la figure 10.

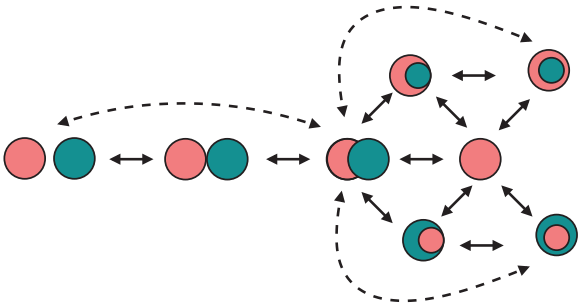


FIG. 10 – Le principe du glissement continu entre deux objets appliqué sur toutes les configurations zone-zone.

Le principe du glissement continu fait de sauts et de glissements simples doit maintenant être appliqué aux objets complexes. La figure 11 est un extrait du graphe d'adjacence entre deux zones complexes. Les deux zones sont d'abord éloignées, leurs périphéries ne se coupent pas. Un saut entre les deux périphéries les amène en position où elles se coupent, approchant ainsi les deux zones. Ensuite un nouveau saut permet à la périphérie de la grande zone de couper la petite zone, approchant à nouveau les deux zones.

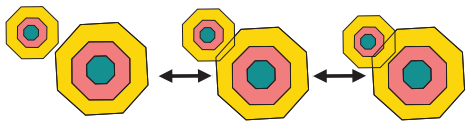


FIG. 11 – Le principe du glissement continu appliqué entre deux zones complexes.

Nous venons de donner un extrait du graphe d'adjacence conceptuelle entre deux zones complexes. Le nombre de configurations y est très grand et son calcul complet n'est pas possible manuellement. Nous avons donc développé un algorithme permettant de calculer ce graphe, c'est cet algorithme que nous allons présenter dans la section qui suit.

3.2 Calcul des glissements sur les objets complexes

Le calcul des glissements sur les objets complexes va s'effectuer comme suit : les deux objets complexes vont être décomposés en objets simples comme pour l'analyse d'une configuration. Dans le cas de deux zones complexes, nous avons alors neuf relations entre objets simples. Pour chacune de ces relations il existe des voisins (glissements simples ou sauts) dans le graphe d'adjacence sur les objets simples. À priori, n'importe lesquels de ces glissements peuvent être appliqués sur une ou plusieurs configurations entre objets simples. Ensuite une règle permet de déterminer si les glissements choisis doivent être retenus, autrement dit si ces glissements respectent le principe du mouvement continu.

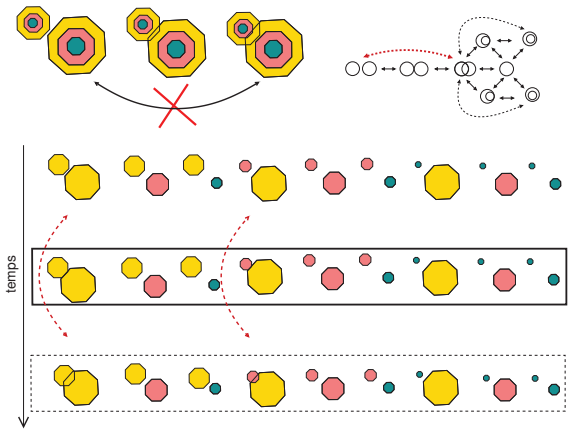


FIG. 12 – Rejet du lien entre les deux configurations.

Sur la figure 12, nous trouvons en haut à gauche les trois configurations extraites du graphe qui nous a servi d'exemple pour spécifier le mouvement continu sur les objets complexes. À droite, nous retrouvons le graphe sur les objets simples. En dessous nous trouvons la décomposition en objets simples de la première configuration du graphe, celle où les deux zones sont le plus éloignées. Deux sauts sur les objets simples sont choisis, ils sont représentés par des flèches, amenant les deux périphéries à se couper ainsi que la périphérie de la grande zone à couper la petite zone (configuration décomposée en bas de la figure). Ces deux sauts nous amènent dans la troisième configuration. Pour passer de la première configuration à la troisième, il est nécessaire de passer par la seconde configuration : donc ces deux sauts ne doivent pas être retenus, d'où la question : qu'est-ce qui nous permet de ne pas retenir ces deux sauts ?

Rappelons le principe du mouvement continu : deux configurations sont liées si on peut passer de l'une à l'autre par un mouvement continu sans passer par une configuration intermédiaire. En ce qui concerne les objets complexes, plusieurs glissements sur les objets simples peuvent être effectués. Comme on ne doit pas passer par une configuration intermédiaire, cela signifie que ces glissements doivent s'effectuer simultanément. Observons sur l'exemple les deux sauts que nous devons donc effectuer simultanément. Un premier saut amène les deux périphéries de la position disjointe vers la position où elles se coupent. Et de même pour le second saut qui amène la périphérie de la grande zone et la petite zone de la position disjointe vers la position où elles se coupent. Ces deux sauts passent par dessus la position intermédiaire dans laquelle les zones se touchent (configuration décomposée du milieu). Cette position intermédiaire correspond à la position à l'instant de la transition, donc si les deux sauts s'effectuent simultanément c'est toute la

configuration intermédiaire qui est la configuration à l'instant de la transition. On peut alors s'apercevoir que cette configuration n'est pas possible topologiquement : la périphérie de la grande zone doit simultanément toucher la périphérie de la petite zone ainsi que la petite zone. De cette façon, ce glissement qui n'est pas souhaité peut être rejeté par cette vérification.

De façon générale, il faut vérifier la cohérence topologique de la configuration complexe à l'instant de la transition (configuration du milieu sur la figure) et après la transition, dans la configuration finale (configuration du bas sur la figure). Sur la figure la configuration topologiquement possible est encadrée en trait discontinu alors que la configuration topologiquement impossible est encadrée en trait continu.

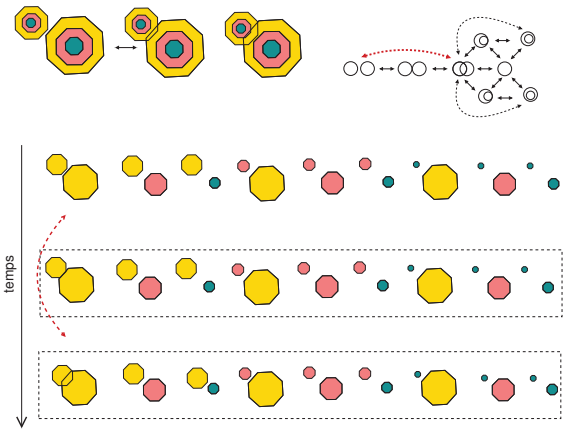


FIG. 13 – Lien accepté entre les deux configurations.

Sur la figure 13, un seul saut a été choisi : le saut amenant les deux périphéries à se couper, ce qui nous amène dans la seconde configuration du graphe. Cette fois-ci, à l'instant de la transition comme après la transition, la configuration est topologiquement possible, le lien est donc créé dans la graphe. Depuis cette seconde configuration, un second saut peut être choisi, voir figure 14. Encore une fois, à l'instant de la transition comme après la transition la configuration est topologiquement possible. Ceci nous permet de créer le second lien dans le graphe qui nous amène à la troisième configurations Nous avons ainsi construit automatiquement le graphe d'adjacence conceptuelle sur les objets complexes en respectant le principe du mouvement continu des objets. Le programme nous permet de dénombrer 3581 nœuds dans le graphe, d'où l'intérêt de posséder un algorithme pour le construire.

Revenons maintenant à notre interface et à l'utilisateur insatisfait de l'interprétation spécifiant que le terrain touchait presque le bois. Par l'intermédiaire de glissements de sens sur les objets complexes nous pouvons lui proposer d'autres choix, voir figure 15, parmi lesquels l'utilisateur trouvera

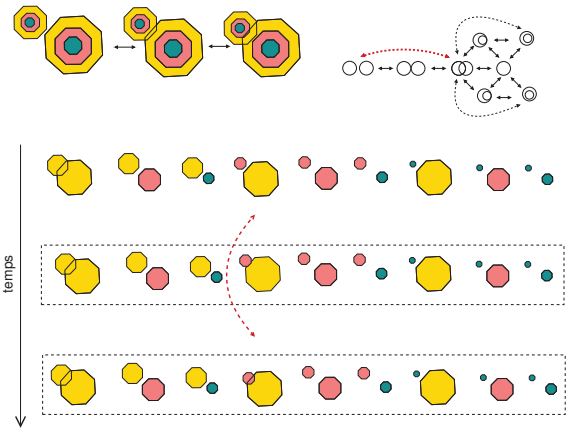


FIG. 14 – Lien accepté entre les deux configurations.

satisfaction. Il aura ainsi transformé la première interprétation pour obtenir celle-ci qui le satisfait :

Peu importe la relation entre la rivière et le bois.
La rivière passe par le bord du terrain.
Le terrain est proche du bois.

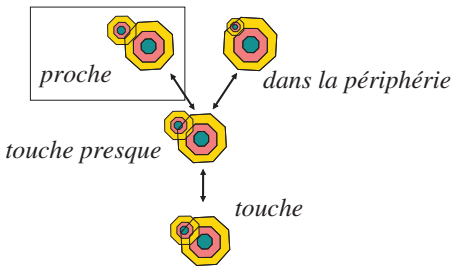


FIG. 15 – Grâce à notre graphe d'adjacence sur les objets complexes nous pouvons proposer à l'utilisateur des interprétations correspondant à des schémas proches du schéma initialement dessiné.

Nous venons ici de présenter le fonctionnement de notre modèle d'analyse spatiale. Ce modèle répond aux besoins que nous avons spécifiés auparavant quant à l'interaction que nous voulions mettre en œuvre entre l'utilisateur et le système. La section qui suit, dernière de cet article, présente la réalisation informatique qui implante sur machine l'algorithme qui vient d'être présenté. Cette réalisation n'est pas encore une interface «grand public» pour l'utilisateur naïf. Elle prend pour l'instant la forme d'un outil permettant l'expérimentation spatio-conceptuelles, nous permettant ainsi de tester le bon fonctionnement des glissements de sens.

4 L'outil et l'expérimentation

Nous nous sommes jusqu'ici attaché au domaine de l'analyse spatiale qualitative. Cette dernière section va porter sur le domaine de la cognition spatiale où le modèle d'analyse spatiale présenté sera vu comme un outil d'expérimentation. Dans une première partie nous présenterons la réalisation informatique : l'outil. Dans une seconde partie nous présenterons l'expérimentation en cours.

4.1 Une réalisation modulaire

La réalisation informatique propose à l'utilisateur de spécifier sa requête sur l'interface de la figure 1. Le système possède une base de données d'un espace restreint, la *Suisse Normande*, une région géographique au sud de Caen appelée ainsi à cause de son relief marqué. Une fois que l'utilisateur a spécifié sa requête, elle est transformée en requête SQL puis envoyée à la base de données. Cette base contenant les informations sur la géolocalisation de différents lieux, la réponse retournée à l'utilisateur est une carte.

L'intérêt du modèle et de la réalisation est sa modularité. Comme nous pouvons le voir sur la figure 16, quatre fichiers permettent de configurer le système. Le premier, Composition, permet de spécifier la taille du centre et de la périphérie des zones ; le second, Étiquetage, permet d'associer un terme à une configuration ; le troisième, Verbalisation, permet de spécifier la façon de verbaliser les termes présents dans le fichier Étiquetage ; enfin le fichier Transformation permet de spécifier comment transformer les termes présents dans le fichier Étiquetage en requête SQL.

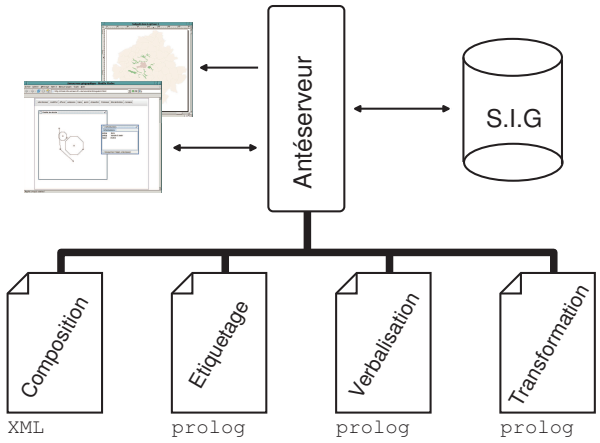


FIG. 16 – La réalisation et ses quatre fichiers de configurations.

Voici un exemple de fichier Composition, celui-ci définit la taille du centre et de la périphérie respectivement égales à 0.25 et 1.25 fois la taille de la zone de référence :

```
<description>
  <entiteComplexe nom="o" type="octogone">
    <entiteSimple id="1" nom=''centre''
                  type="octogone">
      <attribut nom="x" valeur="o.x"/>
      <attribut nom="y" valeur="o.y"/>
      <attribut nom="r" valeur=''o.r*0.25''/>
    </entiteSimple>
    <entiteSimple id="2" nom=''peripherie''
                  type="octogone">
      <attribut nom="x" valeur="o.x"/>
      <attribut nom="y" valeur="o.y"/>
      <attribut nom="r" valeur=''o.r*1.25''/>
    </entiteSimple>
  </entiteComplexe>
</description>
```

Figure 17 nous avons associé huit termes aux huit configurations entre zones simples, voici comment s'écrivent ces termes dans le fichier Étiquetage :

```
interprete_zone_zone_simple(
   [[0,0,1], [0,0,1], [1,1,1]], r1,
   disjoint).
interprete_zone_zone_simple(
   [[0,0,1], [0,1,1], [1,1,1]], r2,
   touche).
interprete_zone_zone_simple(
   [[1,1,1], [1,1,1], [1,1,1]], r3,
   coupe).
   ...
```

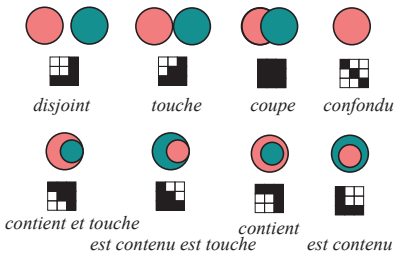


FIG. 17 – L'étiquetage des configurations simples entre deux zones

À chaque fois qu'un nouveau terme est introduit dans le fichier Étiquetage, il faut définir la façon de le verbaliser dans le fichier Verbalisation. Ici les trois termes *disjoint*, *touche* et *coupe*. Le système connaît un lexique de verbes et de pronoms auquel il est possible de faire référence.

```
verbe(disjoint, [[v, "etre"],
                 [a, "disjoint"],
                 [p, "de"]]):-!.
verbe(touche, [[v, "toucher"]]):-!.
verbe(coupe, [[v, "couper"]]):-!.
   ...
```

Ainsi configuré, notre système est déjà utilisable. Prenons l'exemple de la figure 18 sur lequel l'utilisateur a

dessiné deux zones, l'une représente une commune nommée Thurry-Harcourt l'autre représente un bois. Ces deux zones sont éloignées, c'est à dire que leurs périphéries ne se coupent pas. D'après les étiquettes que nous avons attribuées aux configurations, le système attribue le terme *disjoint* au dessin. Comme glissement, le système propose seulement de ne plus tenir compte d'aucune contrainte avec le glissement *peu importe la relation entre la commune et le bois*. En effet, les deux périphéries étant disjointes, la seule configuration voisine est celle où les deux périphéries se coupent. Or dans cette configuration, les deux zones sont toujours disjointes donc il n'y a pas de nouveau terme à attribuer pour cette configuration voisine.

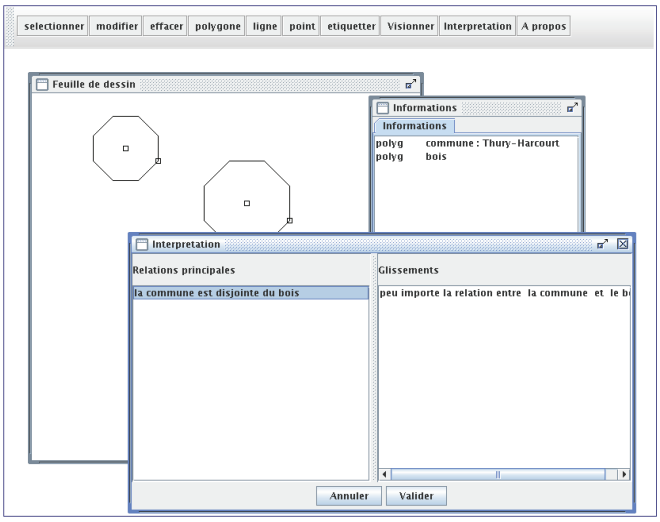


FIG. 18 – Une première utilisation seulement avec les huit étiquettes sur les configurations simples.

Nous pouvons alors définir des configurations ainsi que la façon de les verbaliser pour des objets complexes, ici le terme *proche*, comme sur la figure 20 (`Rfp` signifie relation frontière-périphérie ; `Rpf` : relationpériphérie-frontière ; etc) :

```
interprete_zone_zone_complexe(
   [Rpp, Rpf, _, Rfp, _, _, _, _, _],
  [[Rpp, Rpf, 0, Rfp, 0, 0, 0, 0, 0],
   1, 1, 0, proche]) :-
   interprete_zone_zone_simple(_, Rfp, disjoint),
   interprete_zone_zone_simple(_, Rpf, disjoint),
   interprete_zone_zone_simple(_, Rpp, coupe).

verbe(proche, [[v, "etre"],
               [a, "proche"],
               [p, "de"]]):-!.
```

Avec ce nouveau terme, l'interaction change alors immédiatement, voir figure 19. Avec le même schéma, une modification de la configuration amène les périphéries à se couper, on se trouve alors dans la configuration *proche* que le système propose comme glissement.

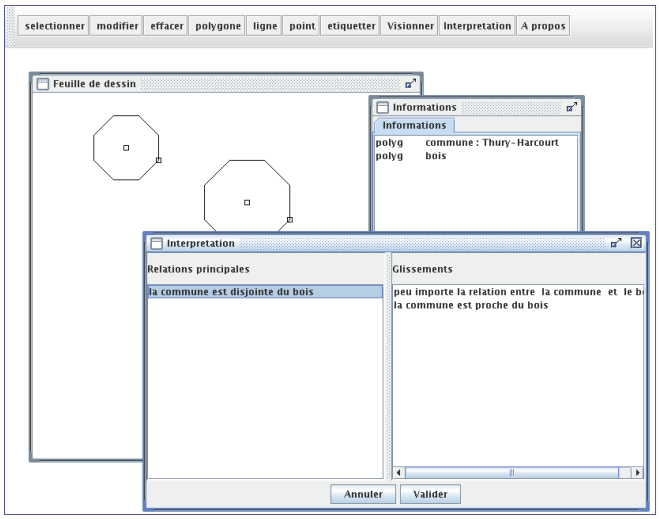
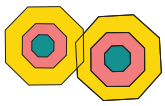


FIG. 19 – Avec une configuration complexe étiquetée, un glissement de sens est alors trouvé.



proche

FIG. 20 – Une étiquette sur une configuration complexe.

Pour chaque terme introduit on doit aussi définir, dans le fichier Transformation, la façon de le transformer en requête SQL. Ici transformation du terme *proche* :

```
sql(O1,O2, proche, S) :- concat_string(
  [
   "Relate(", O1, ".the_geom, ",
              O2, ".the_geom,
              'FFTFFTTTT')
    and
    Distance(centroid(", O1, ".the_geom),
             centroid(", O2, ".the_geom)) < 4500"
   ],S), !.
```

ce qui permet par la suite au système de générer automatiquement la requête SQL suivante :

```
SELECT bois.the_geom, bois.toponyme,
       terrain.the_geom, terrain.toponyme
FROM espacevert bois, constructible terrain
WHERE Relate(bois.the_geom,
             terrain.the_geom,
             'FFTFFTTTT')
    and
    Distance(centroid(bois.the_geom),
             centroid(terrain.the_geom)) < 4500
```

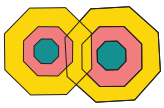
Grâce à ces règles de transformation, le système est capable

d'interroger une base de données et de retourner l'information à l'utilisateur sous forme de carte.

Comme nous pouvons le constater le système est particulièrement flexible permettant facilement de dimensionner les objets complexes et d'attribuer des termes de notre choix à des configurations. En amont le système agit comme un moteur qui analyse le schéma pour lui attribuer des termes le caractérisant, calcule des configurations voisines du schéma et attribue des termes à ces configurations. Dés qu'un nouveau terme est spécifié, il sera immédiatement pris en compte pour caractériser une configuration, qu'elle soit dessinée par l'utilisateur ou bien voisine de celle-ci. Grâce à cette flexibilité, nous souhaitons utiliser cette réalisation afin de réaliser une expérimentation sur des utilisateurs.

4.2 L'expérimentation

Le système est capable de proposer des termes caractérisant non pas le schéma dessiné mais un schéma ressemblant. Ceci doit nous permettre d'associer un terme T à une configuration complexe C très précise. Si on demande à un utilisateur de dessiner un schéma correspondant au terme T, il y aura alors peu de chance qu'il dessine un schéma exactement dans la configuration C, mais il doit y avoir de fortes chances pour que son schéma soit dans une configuration ressemblante, appelons la C'. Ainsi, par l'intermédiaire d'un glissement de sens, le système doit proposer à l'utilisateur le terme T correspondant à la configuration C·



très proche

FIG. 21 – Une nouvelle étiquette sur une configuration complexe.

Prenons un exemple et étiquetons la configuration de la figure 21 par le terme T = *très proche* correspondant à la configuration C où toutes les relations simples sont spécifiées :

```
interprete_zone_zone_complexe(
   [Rpp, Rpf, Rpc, Rfp, Rff, Rfc, Rcp, Rcf, Rcc],
  [[Rpp, Rpf, Rpc, Rfp, Rff, Rfc, Rcp, Rcf, Rcc],
   1, 1, 0, tres_proche]) :-
   interprete_zone_zone_simple(_, Rpp, coupe),
   interprete_zone_zone_simple(_, Rpf, coupe),
   interprete_zone_zone_simple(_, Rpc, disjoint),
   interprete_zone_zone_simple(_, Rfp, coupe),
   interprete_zone_zone_simple(_, Rff, disjoint),
   interprete_zone_zone_simple(_, Rfc, disjoint),
   interprete_zone_zone_simple(_, Rcp, disjoint),
   interprete_zone_zone_simple(_, Rcf, disjoint),
   interprete_zone_zone_simple(_, Rcc, disjoint).
```

Nous allons demander à un échantillon d'utilisateur de dessiner, à l'aide de l'interface, deux zones très proche l'une de l'autre. Nous verrons quel pourcentage à réalisé un schéma exactement dans la configuration C ou dans une configuration C' telle qu'il existe un lien dans le graphe entre C et C'. En fonction des résultats obtenus, nous pourrons faire différentes expériences en modifiant le terme associé à la configuration ou en modifiant la dimension de la périphérie ou du centre de nos zones. D'autres configurations seront bien entendu essayées.

5 Conclusion

Ce projet d'interface relève de deux domaines. Comme c'est une interface pour des utilisateurs, il fait partie du domaine de l'interaction utilisateur-système. En raison de son mode de fonctionnement, il fait aussi partie du domaine de l'analyse spatiale qualitative. Le travail a jusqu'ici porté sur le second domaine : l'analyse spatiale. Ce travail a abouti à une réalisation informatique qui met à notre disposition un outil d'expérimentation spatio-conceptuelle. Un premier projet d'expérimentation a d'ailleurs été décrit dans cet article. Elle nous permet de soumettre l'utilisateur à une première utilisation très guidée de l'interface. C'est ainsi que nous allons déplacer notre domaine de recherche de l'analyse spatiale vers l'interaction utilisateur-système pour que notre système puisse être utilisé par tout public.

Références

[1] F. Dumoncel, C. Bazin, M. Gaio et J. Madelaine. Bimodal interaction loop for a geo-spatial query. In *2005 HCI International, 11th International Conference on Human-Computer Interaction*, Las Vegas, 2005.

[2] F. Dumoncel, M. Ould Ahmed Limam et M. Gaio. Feedback sémantique dans le cadre d'un environnement de recherche d'informations géographiques. In *IHM 2003*, pages 88–95, Caen, France, novembre 2003. ACM.

[3] M. Egenhofer. Reasoning about binary topological relationships. In O. Günter et H.-J. Schek, éditeurs, *Advances in Spatial Databases, SSD'91. Lectures Notes in Computer Science*, volume 525, pages 143–160. Springer-Verlag, 1991.

[4] M. Egenhofer, J. Sharma et D. Mark. A critical comparison of the 4-intersection and 9-intersection models for spatial relations : Formal analysis. pages 1–11. R. McMaster and M. Armstrong, 1993.

[5] C. Freksa. Conceptual neighborhood and its role in temporal and spatial reasoning. In *Decision Support Systems and Qualitative Reasoning*, pages 181–187. 1991.

[6] Cohn A . G., B. Bennett, J. Gooday et N. M. Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3) :275–316, 1997.

[7] M. Ould Ahmed Limam. *Interaction avec feedback sémantique dans un environnement dédié à la recherche d'informations géographiques*. PhD thesis, Université de Caen, Caen, Septembre 2003.

[8] M. Szmurlo et M. Gaio. Extended conceptual neighborhoods. In D. Fritsch, M. English et M. Sester, éditeurs, *Proc. of the ISPRS Comm. IV Symposium : Gis - Between Visions and Applications*, volume 32. Institute for Photogrammetry, Stuttgart University, 7-10 September 1998.

Gestion du temps dans la programmation fonctionnelle paresseuse

Jerzy Karczmarczuk

Dept. d'Informatique, Université de Caen,
Sciences 3, Bd. Maréchal Juin, 14032, Caen
karczma@info.unicaen.fr

Résumé Nous analysons la programmation des processus itératifs, naturels dans les calculs scientifiques et dans la simulation, vue de la perspective de programmation fonctionnelle non-stricte (paresseuse), qui offre une approche *statique* aux constructions codées et permet de raisonner sur le rôle du temps dans le programme de manière très souple, parfois non-orthodoxe. Les exemples concernent la construction des flots infinis, co-récursifs, et des modèles non-triviaux de l'enchaînement des états d'un programme fonctionnel. Nos structures de données référencent les éléments « futurs », pas encore construits, et la séquence des états peut paraître comme ordonnée contrairement à la flèche du temps.

Mots-clés : Haskell, programmation paresseuse, états, différentiation.

1 Introduction. Causalité vs. téléologie

Dans le calcul et dans la programmation dite : « scientifique » : processus itératifs (par exemple la solution des équations différentielles), simulation événementielle, etc., la notion de temps est intuitive, et souvent implicite. La notion de causalité classique est bien intégrée dans notre méthodologie de travail, et on n'y pense plus, elle est naturelle.

Le temps est incrémenté afin de passer au point suivant d'une trajectoire physique, les *états* du système se succèdent de manière causale, les instructions de notre programme s'exécutent séquentiellement, en suivant cette « flèche du temps ». Que peut-on ajouter à cela, sauf un peu de formalisation pas forcément fascinante ?...

Or, vu de la perspective globale, le déroulement temporel de quelques processus programmés n'est pas si trivial que cela. En particulier, parfois on est obligé de composer avec le comportement *finaliste*, ou *téléologique* (***goal-oriented***). Le programme doit « sentir le futur » (bien sûr, c'est une manière figuratif de parler).

– Dans la robotique ou dans les techniques d'animation, on exploite la technique de *cinématique inverse* [2]. Le programme sachant quelle trajectoire doit suivre le bout de l'effecteur (la main) d'un robot ou d'un personnage animé, recalcule les forces appliquées aux articulations du bras, de l'épaule et d'autres parties du corps, afin d'engendrer les mouvements intermédiaires de l'épaule etc. Ceci est évidemment non-causal, car mélange le mouvement provoqué par les lois dynamiques et cinématiques avec un *scénario* établi *a priori*.

– Dans le domaine de compilation, un analyseur qui construit l'arbre syntaxique par la méthode ascendante, et qui récupère les items lexicaux : feuilles, lie ces feuilles en expressions, fusionne les expressions, etc., afin d'arriver à la racine de l'arbre, qui représente le programme entier, propage/synthétise les attributs (propriétés sémantiques) des nœuds intermédiaires de l'arbre syntaxique depuis les feuilles vers la racine.
Or, il existe aussi la notion d'*attributs hérités* (souvent : contextuels, par ex. les précédences) qui « descendent » de la racine vers les feuilles. La gestion de ces attributs demande une sorte de raisonnement « anti-temporel » [1] et rend le codage des parseurs ascendants un peu délicat, souvent les attributs hérités sont interdits.

– La technique de *différentiation automatique inverse*, capable de calculer de manière numérique, mais *algébrique et exacte* la dérivée d'une fonction $f(x)$ codée comme une procédure numérique [3, 4], construit pour toute variable interne v son *adjoint* $\bar{v} = \partial f/\partial v$ dans le sens contraire vis-à-vis du « temps » du programme, d'abord $\bar{f} = 1$, et termine par $\bar{x}$.
Puisque f n'est connu qu'à la fin du programme, la technique typique consiste à exécuter la procédure en accumulant des données essentielles pour l'algorithme dans une liste séquentielle, et à la fin du parcours, renverser le parcours et reculer vers le début, là où x est vu pour la première fois. De tels programmes opèrent avec « deux flèches temporelles » simultanément.

– Un réseau neuronal multi-couche qui subit le processus d'apprentissage par la retro-propagation des erreurs, est un autre exemple de système qui « marche contre la flèche du temps » [5, 6]. La mise à jour des poids synaptiques lors de la phase d'apprentissage procède de la couche de sortie vers les entrées du réseau ; c'est une solution technique parfaitement légitime, et souvent exploitée, mais contraire à notre intuition concernant le fonctionnement d'un réseau physique...

Tout ceci demande l'implémentation d'un flot d'information *antithétique* par rapport à la transmission de l'information causale. Il existe aussi un exemple plus simple d'un tel phénomène : le retour en arrière (*backtracking*) dans des programmes logiques. Si un problème en Prolog admet plusieurs solutions alternatives (plusieurs pas possibles dans un algorithme de recherche ou un jeu), on peut demander à ce que une solution partielle soit oubliée, le programme « anéantit le temps » et recule jusqu'à un point de branchement, où une autre décision aurait lieu. Le temps « bifurque »...

Donc, les couches d'abstraction de la programmation moderne disposent de quelques dispositifs techniques permettant de nous libérer de contraintes temporelles naturelles, et de considérer le temps simplement comme un paramètre qui peut être manipulé comme en mathématiques, où l'ordre n'est qu'une relation formelle. Cette vision statique du temps est **la** conceptualisation exploitée par les physiciens théoriciens depuis 3 siècles, et philosophiquement elle a gagné en influence avec l'arrivée de la théorie de relativité, qui a un peu « désacralisé » le temps. Peut-on en profiter dans la programmation scientifique de manière élégante et cohérente ?

La programmation logique non-déterministe, avec *backtracking* en est une possibilité populaire, mais demande l'usage d'un style de programmation très particulier, artificiel.

Dans cette présentation nous allons utiliser les techniques de *programmation fonctionnelle paresseuse*, où l'exécution d'une procédure f appliquée à l'expression e ne commence pas par l'évaluation de e. L'expression est transmise à la procédure sous forme latente, différée : c'est une référence vers le code qui calculerait e. Seulement quand f ***demande*** la valeur de e pour pouvoir progresser dans le calcul, l'argument est évalué/exécuté, et l'objet fonctionnel anonyme qui représentait cet expression, est remplacé par la valeur finale. (Les accès ultérieures utilisent la valeur finale : on appelle ce protocole de passage des paramètres : *call by need*.)

Dans un programme qui suit ce protocole, le temps se « désintègre », on ne peut *a priori* dire quand une expression sera évaluée, quand une variable sera instantiée, etc. On doit garder une vision globale, statique de la totalité du processus.

2 Quelques algorithmes paresseux

2.1 Traitement des listes

La lecture de cet article demande un minimum de connaissances de la programmation fonctionnelle, l'essentiel du lambda-calcul, etc. Nous allons utiliser le langage paresseux Haskell [8] qui permet l'écriture de programmes très compacts. Dans ce langage, l'opérateur `( :)` est l'équivalent « `cons` » du Lisp, il permet d'assembler les listes : si `l=[1,2]`, alors `7 :l` est la liste `[7,1,2]`. Voici une procédure qui construit une liste infinie d'entiers : `[0,1,2,3,...]` :

```
entiers = arithseq 0 where
 arithseq n = n : arithseq (n+1)
```

C'est une forme récursive, sans fin, sans le cas de base, une « fuite en avant » qui est parfaitement légitime et pratique grâce au fait que le résultat – la liste `entiers` n'a pas besoin de s'instantier complètement. Son premier élément est donné explicitement, mais le second est engendré uniquement si on demande sa valeur. Alors il est créé, et les éléments suivants restent différés, jusqu'à leur instantiation *par le consommateur*. On peut aussi transmettre et stocker ailleurs une valeur différée sans sa « dévirtualisation ».

Le parcours par un segment fini d'une telle liste remplace une boucle `for` dans d'autres langages, un processus itératif. D'ailleurs, le Prélude Standard de Haskell contient déjà cette définition, accessible par la syntaxe `entiers = [0 ..]`. De tels algorithmes portent le nom de ***co-récursifs***.

Haskell permet de surcharger les opérateurs, par exemple l'addition de deux listes, élément par élément, et dispose de quelques dispositifs syntaxiques fort utiles. Les paires : tête `:` queue ou autres *constructions* peuvent figurer dans la liste des paramètres des procédures, ce qui facilite leur dé-structuration et reconnaissance (et évite l'usage trop fréquent des sélecteurs (comme `head` et `tail`) omniprésents en Lisp).

Ceci permet des constructions paresseuses encore plus étonnantes, qui seront présentés de manière très simplifiée, car la gestion de la surcharge en Haskell demande l'usage des *classes*, qui ne seront pas discutées ici.

Voici une autre définition de la liste de tous les entiers naturels. Ici `uns` est la liste infinie de 1.

```
(x:xq) + (y:yq) = (x+y) : (xq+yq)
uns = 1 : uns

entiers = 0 : (entiers + uns)
```

En effet, le second élément de `entiers` est la somme de 1 et son premier élément qui est égal explicitement à zéro. Ceci permet le calcul de son troisième élément, etc., à condition que la liste soit parcourue séquentiellement.

Récapitulons le résultat conceptuel de cet exercice : nous avons implémenté une *structure de données*, qui représente un *processus*, même infini (sa définition ne contient pas des bornes). Bien sûr, pour instantier ce processus, pour le rendre dynamique, il faut parcourir la liste, mais l'algorithmique de sa création est statique. La programmation et le raisonnement deviennent plus faciles.

Bien sûr, il est facile aussi de créer des pathologies. Les deux exemples ci-dessous déclenchent, lors de la tentative d'usage des variables `r` ou `u` des exceptions de bouclage :

```
r=x where x=x
u=x where x=1+x
```

car *aucune partie de x* ne peut être calculée sans connaître x, tandis que dans `x=1:x` – si.

2.2 Génération du son

La méthode économique de génération d'une sinusoïde $\sin(x), \sin(x+h), \sin(x+2h), \ldots$, qui évite des applications itérées des fonctions trigonométriques, est basée sur la récurrence :

$$\sin(x+2h) = 2\cos(h) \cdot \sin(x+h) - \sin(x) \qquad (1)$$

ce qui permet de créer incrémentalement un flot de valeurs qui sera interprété comme l'amplitude d'une onde monochromatique échantillonnée, et qui peut être temporisé et envoyé au convertisseur acoustique. Classiquement on construit des valeurs successives de ce flot dans une boucle. Peut-on le définir d'un seul coup ? Voici la liste paresseuse qui l'implémente [9], en commençant par $x = x_0$.

```
sins = sin x0 : q where
   q = sin(x0+h) : (2*cos h *> q - sins)
```

où `*>` est l'opérateur qui multiplie une liste (tous ses éléments) par un scalaire :

```
a*>(x:xq) = a*x : a*>xq
```

Une liste qui « se mange la queue » peut être également utilisée dans l'algorithme de Karplus-Strong [10] qui simule une corde pincée (guitarre, clavecin, etc.). C'est un algorithme cyclique qui initialise un tampon séquentiel avec des valeurs aléatoires (bruit blanc), et ensuite fait passer les données par un filtre passe-bas, en recyclant le même tampon, et en dérivant le flot de données vers la sortie. Le fitrage élimine vite toutes les fréquences, sauf quelques fondamentales, qui dépendent de la période – la longueur du tampon. Après avoir choisi la période (inversement proportionelle à la fréquence fondamentale de la corde), il suffit d'écrire

```
bing = (whitenoise periode) ++
         0.5 *> (bing + tail bing)
```

où `++` concatène les listes. Tout le filtrage (ici très simple) : $y_n = 1/2 \cdot (x_n + x_{n-1})$ se réduit au calcul de la moyenne entre la liste et de la même liste décalée d'un élément, mais les fitres plus élaborés, y compris les fitres récurrentiels (IR) peuvent être codés de manière aussi compacte. La fonction `whitenoise` est un générateur de séquence pseudo-aléatoire de longueur donnée.

L'article [9] présente d'autres simulations des instruments musicaux, basées sur la modélisation physique, la circulation des ondes sonores dans des cordes et cavités.

2.3 Équations différentielles

Nous montrerons comment coder un algorithme de solution numérique (avec l'incrément h fini) de l'équation $x' = f(x,t)$ avec $x(t=t_0) = x_0$ de manière à ce qu'il engendre la trajectoire complète, potentiellement infinie. Pour simplicité présentons l'algorithme scalaire de Runge-Kutta d'ordre 2, mais ceci peut être généralisé facilement. L'organisation habituelle de l'algorithme est la suivante :

$$\begin{aligned} k_1 &= h \cdot f(x_n, t_n)\,, \\ k_2 &= h \cdot f(x_n + k_1/2, t_n + h/2)\,, \\ x_{n+1} &= x_n + k_2\,. \end{aligned} \qquad (2)$$

Si la fonction f est adaptée aux listes, ou écrite de manière suffisamment flexible, et ainsi peut être appliquée à tous les objets qui admettent les opérations arithmétiques convenablement surchargées, listes incluses, alors la trajectoire complète s'exprime par la formule suivante [11] :

```
t=[t0,t0+h ..]
x=x0 : x + h*>f (x+0.5*>k1)(t+(0.5*h)*>uns)
  where k1 = h*>f x t
```

Le code est très compact, l'utilisateur a moins d'occasions de commetre des fautes. On sépare la *génération* de la trajectoire du contexte de son utilisation, donc on a pas besoin d'incorporer les conditions d'arrêt de la boucle dans le générateur, ceci doit être pris en charge par le consommateur. On est un peu plus le maître du temps que son esclave…

Pour terminer cette section avec un commentaire général : le monde scientifique et technique utilise des techniques présentées ici plutôt rarement, le mythe de l'inefficacité des langages fonctionnels étant toujours présent dans les esprits. Cependant des langages comme Haskell ou Clean (ou le langage strict (non-paresseux) OCaml) deviennent compétitifs, et indépendamment des applications professionnelles ils offrent un *style* bien adapté à la *pédagogie de programmation*.

3 Programmation fonctionnelle et transition d'états

3.1 Explicitation des états

La notion d'état du système dans le monde de programmation impérative traditionnelle est souvent implicite. Le programme dispose d'un certain nombre de variables globales qui sont mis à jour (les effets de bord) par les procédures ; leurs valeurs, ainsi que le contenu des tampons d'entrée/sortie ensemble constituent l'état. (Les entrées et sorties en général constituent un problème plus complexe, car ici le programme peut changer l'état du « monde extérieur », le disque ou le réseau. cependant les paradigmes paresseux en Haskell s'avèrent également utiles dans ce contexte.)

Dans la programmation fonctionnelle il n'y a pas d'effets de bord, c'est un modèle de calcul qui s'approche au raisonnement mathématique, donc rien n'est caché. En principe on n'a pas le droit de modifier sur place une variable. Alors, si nous voulons qu'une action du programme : une opération arithmétique, un appel de procédure, etc., produise un *effet* spécial, par exemple l'incrémentation d'un compteur global, il faut que cet état-compteur soit *explicite*.

On pourrait procéder de manière suivante : au lieu d'opérer avec les valeurs, disons x, auxquelles on applique une fonction : $f : x \to f(x)$, on construit des *paires* (x, s), où s est une structure de données qui symbolise l'état. Si s est le compteur, l'application (convenablement généralisée) de la fonction f doit engendrer l'objet $(f(x), s + 1)$.

Cependant, cette stratégie n'est pas si simple, puisque l'on ne peut faire proliférer les compteurs partout, le programme doit assurer que l'état du système existe en un seul exemplaire, un seul compteur accessible à un moment donné. On **peut** imaginer et implémenter non pas un compteur mis à jour, mais une séquence de compteurs, où chaque nouveau remplace l'ancien, avec une nouvelle valeur, cependant la création d'un nouveau compteur doit anéantir la possiblité d'accès aux précédents.

Ceci changera notre vision des structures de données, une valeur x (par ex. réelle) sera étendue vers une entité fonctionnelle, $\lambda s \to (x, s)$ (en Haskell : `\s->(x,s)`). C'est une fonction *qui agit sur l'état* et ici – seulement le répertorie, sans changement.

Répétons l'essentiel : tandis que dans la programmation impérative les opérations sur les données peuvent changer l'état du système, dans le modèle fonctionnel les données elles-mêmes, les valeurs des expressions obtiennent le status des actions. Elles sont « liftées » vers le monde fonctionnel.

Une fonction primitive f agissant sur x provoque aussi le changement d'état, elle prend donc deux arguments ; en général toute *expression* devient une fonction agissant sur l'état initial, et qui retourne une valeur et l'état final, égal au initial, ou éventuellement modifié. En Haskell nous pouvons définir un opérateur `(>=>)` tel, que `f >=> m` exécute l'expression `m` sur l'état initial, produisant l'état intermédiaire. La fonction `f` agit sur la valeur et cet état, et modifie – éventuellement – les deux, où le mot *modification* signifie la *construction* des nouvelles valeurs. Les anciennes peuvent être oubliées. Voici la définition de cette opération, assez courte :

```
f>=>m=\s_init->let (x,s_mid)=m s_init
                   (y,s_final)=(f x) s_mid
                in (y,s_final)
```

Si classiquement on applicait séquentiellement $p = f(g(x_0))$ à une valeur, dans le nouveau modèle nous aurions

```
m0 = \s->(x0,s)
p = f >=> g >=> m0
```

où, par exemple

```
f x cmpt = (sin x, cmpt+1)
```

etc., pour `g`. Toutes les opérations sont chaînées séquentiellement, et le programme devient une énorme fonction, ce qui semble peu commode, mais le modèle est parfaitement utilisable. Après tout, vu de la perspective du système d'exploitation tout programme en train d'exécution peut être assimilé à une énorme fonction, si on considère que l'exécution séquentielle des instructions est due à l'application d'un opérateur spécial : 'séquence'…

(L'exemple ci-dessus ne montre que l'application des opérateurs unaires, ce qui est assez pauvre. Un programme pratique a besoin des opérateurs arithmétiques binaires, procédures à plusieurs arguments, etc. Donc la discussion détaillée serait un peu plus compliquée, mais l'idée de base reste.)

Il faut comprendre ce que nous avons obtenu. `p` est une *expression* dans le nouveau modèle, alors c'est un objet fonctionnel, et il faut qu'il soit appliqué à un état initial afin de faire quoi que ce soit d'effectif. Donc, le « programme principal » serait

```
main = p cmpt_initial where
          cmpt_initial = ...
          p = ...
          ...
          f = ... --  autres définitions
```

Des milliers de programmes en Haskell et en Clean : des gestionnaires de bases de données, programmes numériques, jeux, etc., ont été écrits de cette manière, c'est une technologie maîtrisée. Elle n'a même pas besoin de l'évaluation paresseuse. Il existe une formalisation dite **monadique** de ce style de programmation, exploitée également dans la spécification des entrées et sorties en Haskell, mais nous ne pouvons nous arrêter là. La gestion des états présentée dans la section suivante nécessite la paresse.

3.2 Machine à « reculer dans le temps »

À présent nous allons « pervertir » la gestion des états dans le cadre de la programmation paresseuse. L'idée est basée sur un travail de Wadler [12]. Puisque toute opération dans un programme fonctionnel est une transformation de données, la manipulation des états peut être quelconque, et ne pas suivre l'intuition.

Il nous faudra imaginer que l'expression composite `f >=> m` soit une fonction qui agit sur l'état final, non pas initial. Dans cet état final la fonction f s'applique « normalement » à une valeur obtenue par l'évaluation de m.

Mais, en agissant ainsi, la fonction f produit accessoirement l'état intermédiaire, et c'est à cet état intermédiaire qu'il faut appliquer m pour récupérer sa valeur. L'expression m produit l'état initial.

Ceci ne peut vraiment être intuitivement compris car l'ordre temporel semble être violé, la signification des termes *initial* et *final* n'est plus habituelle, mais on peut l'implémenter en Haskell de manière naturelle :

```
f>=>m =\s_final->let (x,s_init)=m s_mid
                     (y,s_mid)=(f x) s_final
                 in  (y,s_init)
```

En tant que programme, ceci est correct et légitime. Le seul problème est de donner du sens sémantique (pas forcément évident et intuitif) à cette construction, et élaborer quelques exemples d'utilisation pratique.

Mais d'abord regardons la structure de cette définition. On voit clairement une récurrence cyclique : afin d'obtenir `x`, l'expression `m` doit agir sur `s_mid`, mais cet état ne peut être construit que si `f` s'applique à `x` (et à `s_final`). La programmation paresseuse ne fait pas des miracles, et ne viole pas le sens commun. Cette construction peut marcher uniquement si l'évaluation d'un ou plusieurs arguments est différé, si – par exemple – l'expression `m` peut commencer son travail sans avoir besoin de son argument-état. Les états ne seront instantiés qu'à la fin du traitement des données `x`, etc.

4 Différentiation automatique

4.1 Introduction ; méthode directe

Le calcul des dérivées ne se réduit pas aux procédés symboliques ou aux approximations numériques par des différences finies. Il existe une méthode dite « automatique » ou algorithmique (noms historiques…) qui est purement numérique, mais exacte. Elle est adapé à la différentiation des *programmes numériques*, et elle sera présentée ici de manière très superficielle.

Tout programme – après avoir pris les décisions (branchements), et après avoir aplati les boucles, se réduit à une composition fonctionnelle, l'enchaînement des constructions des variables intermédiaires comme expressions qui dépendent des variables déjà définies.

Si le langage de programmation permet la surcharge des opérations arithmétiques, afin d'implémenter la *la méthode directe* dans le style fonctionnel [13], nous allons étendre le domaine numérique (des nombres flottants) vers *des paires* de nombres. Pour simplicité, seulement le cas d'une variable indépendante sera discuté. Toute expression numérique e devient (e, e'), où le second élément représente la dérivée. Les **constantes** c dans le programme prendront la forme $(c, 0)$, et **la** variable indépendante, disons x (qui peut être le nom du paramètre de la fonction dont la dérivée doit être calculée) sera convertie en $(x, 1)$.

Prenons le programme

$$y = \sin(x); \qquad z = y^2 - x/y\,. \tag{3}$$

Il nous suffit de définir l'arithmétique étendue : $(e, e') + (f, f') = (e+f, e'+f')$, $(e, e') \cdot (f, f') = (e \cdot f, e'f + ef')$, $\exp(e, e') = (f, e'f)$ **où** $f = \exp(e)$, etc. pour d'autres fonctions élémentaires, en accord avec des formules scolaires. Le programme devient
$(y, y') = (\sin(x), \cos(x) \cdot x')$;
$(z, z') = (y^2 - x/y, -1/y \cdot x' + (2 \cdot y + x/y^2) \cdot y')$

La compilation du programme étendu peut être automatisée, même si cela demande un peu d'effort.

En général, pour une fonction arbitraire f : $f(e, e') = (f(e), e' \cdot f'(e))$. Une affectation dans le programme : $g \leftarrow f(e_1, e_2, \ldots, e_k)$ engendre l'instruction

$$\frac{dg}{dx} = \frac{\partial f}{\partial e_1}\frac{de_1}{dx} + \frac{\partial f}{\partial e_2}\frac{de_2}{dx} + \cdots + \frac{\partial f}{\partial e_k}\frac{de_k}{dx} \tag{4}$$

et, sachant que les sous-expressions e_i ont déjà été construites avant, si leurs dérivées sont connues également et la fonction f est connue, toute expression g est algorithmiquement calculable avec sa dérivée. C'est une technologie qui devient de plus en plus indispensable dans l'ingénierie.

La méthode directe n'a pas de rapport direct avec le sujet principal de cet article, mais il fallait l'introduire afin de savoir de quoi on parle. Ceci dit, dans [13] nous avons profité du paradigme d'évaluation paresseuse pour construire des listes infinies représentant des expressions numériques avec *toutes* leurs dérivées, d'ordre quelconque, et nous avons montré comment transformer quelques équations récurrentielles non-triviales en algorithmes effectivement implémentables, en utilisant les astuces similaires à celles montrées auparavant.

4.2 Méthode inverse

Dans le cas multi-dimensionnel il faut introduire un gradient pour chaque variable dans le programme, et les expressions deviennent des vecteurs à nombreuses composantes, ce qui n'est pas commode. Dans quelques circonstances, surtout quand on a besoin d'un seul résultat qui dépend de plusieurs variables, on peut procéder différemment, pour l'efficacité on n'introduit pas des gradients, par contre, pour chaque variable v, indépendante ou intermédiaire, on introduit son ***adjoint*** $\overline{v}$. L'adjoint de v est spécifié formellement comme $\overline{v} = \partial z/\partial v$, où z dénote le résultat final du programme.

Mais alors, on ne peut pas facilement étendre l'arithmétique des expressions et convertir, disons, e en $(e, \overline{e})$ au milieu du programme, car avant sa terminaison et avant la construction de z, la valeur de $\overline{e}$ ne peut être connue ! Donc, la valeur de l'adjoint $\overline{x}$ de la variable indépendante (ici également, par simplicité nous traiterons le cas 1-dimensionnel) qui est la dérivé désirée de z : $\overline{x} = dz/dx$ ne pourrait être obtenue qu'à la fin du programme de base. Seulement alors la construction des dérivées procède par l'exécution des *instructions adjointes*.

L'enchaînement des dérivées se fait « à l'envers », ce qui sera expliqué ci-dessous. La technique n'est pas facile à implémenter, mais elle est maîtrisée aussi, voir par ex. [14]. On peut effectuer deux passes, en prévoyant une zone de stockage (une « bande » dans le jargon du domaine) pour les instructions adjointes *accumulées* pendant la première phase du programme, quand on ne calcule effectivement que les valeurs principales.

En général, pour toute affectation d'une variable intermédiaire $g \leftarrow f(e_1, e_2, \ldots, e_k)$ dans le programme, on construit les instructions adjointes, qui ***modifient les adjoints des variables à droite*** :

$$\overline{e}_j \leftarrow \overline{e}_j + \overline{g}\frac{\partial f}{\partial e_j} \tag{5}$$

Soulignons le fait que c'est le premier **usage**, non pas la définition d'un expression e qui définit son adjoint.

Dans notre exemple (3), nous aurons

$$\begin{aligned} z = y^2 - x/y; \quad &\rightarrow \quad \overline{x} \leftarrow \overline{x} + \overline{z}(-1/y); \\ &\quad\ \overline{y} \leftarrow \overline{y} + \overline{z}(2y + x/y^2); \\ y = \sin(x); \quad &\rightarrow \quad \overline{x} \leftarrow \overline{x} + \overline{y}\cos(x)\,. \end{aligned} \tag{6}$$

Finalement l'adjoint de x, la valeur de dz/dx devient $\overline{x} = -1/\sin(x) + \cos(x)\left(2\sin(x) + x/\sin(x)^2\right)$.

Notre idée était d'implémenter cet algorithme en ***une passe*** [15], à l'aide de puissantes techniques sémantiques de la programmation fonctionnelle paresseuse. La section suivante en présente la réalisation.

4.3 Implémentation fonctionnelle

Le lecteur a pu déjà deviner que la phase adjointe de cet algorithme est justement une phase traitée comme antithétique, où le « le temps recule » (ou, plutôt, le fil d'Ariane est embobiné). Il nous reste de spécifier « l'état » dans le programme de façon à permettre au programme de calculer les adjoints durant la seconde phase, qui dans la réalité se déroule simultanément avec la première, où les expressions normales sont évaluées.

Le rapport entre les états et les adjoints est le suivant. Au moment du démarrage du programme, quand on évalue x, l'état initial est constitué de $\overline{x}$. À la fin, en construisant le résultat z, on considère que l'état final soit $\overline{z}$. On constate :

- L'état initial $\overline{x}$ **ne peut** être connu au début des calculs, mais ceci n'est guère gênant, car on n'en aura besoin qu'à la fin. Il peut rester latent, différé, sous forme d'un objet paresseux, non-évalué. C'est une « promesse » de livrer la valeur actuelle, quand toutes les dépendances entre variables seront résolues.
- Par contre, l'état final $\overline{z} = \partial z/\partial z \equiv 1$ est trivialement connu à l'avance, et il peut être utilisé à n'importe quel moment.

Il nous faut à présent combiner l'algèbre des dérivations mathématiques avec la machine antithétique de Wadler, modifiée selon nos besoins. Les expressions seront converties en *fonctions – transformateurs des états*. Leur type aura le type `Ld (a->(a,a))`, où la balise `Ld` est là uniquement pour guider les yeux (une expression **est** une fonction), et `a` dénote le type numérique utilisé, d'habitude les flottants : `Double`, mais les valeurs des expressions peuvent aussi être complexes.

Pour rendre la notation plus conviviale, on n'utilisera aucun opérateur `>=>` explicite, mais tout sera réalisé par des opérations arithmétiques de forme classique, utilisables par un utilisateur quelconque, qui n'a même pas besoin de savoir que ces opérations prennent en charge les dépendances croisées entre données. Le code qui suit peut être placé dans le système de *classes de types* de Haskell, mis dans un module-librairie utilisateur, et compilé normalement. L'utilisateur aura seulement besoin de savoir que ses données de forme `(Ld ...)` sont des entités arithmétiques qui admettent les opérations arithmétiques non-standard.

Les constantes numériques c et les variables x dans un programme augmenté doivent être « liftées » avec des fonctions ci-dessous :

```
lCnst c = Ld (\n->(c,0))
 -- Une constante est triviale
lDvar x = Ld (\n->(x,n))
 -- L'état est multiplié par 1
```

Voici quelques opérateurs arithmétiques qui réalisent les opérations sur les expressions liftées :

```
(Ld pp)+(Ld qq) = Ld (\n->
 let (p,pb)=pp n
     (q,qb)=qq n
 in (p+q,pb+qb))

(Ld pp)*(Ld qq) = Ld (\n->
 let (p,pb)=pp(n*q)
     (q,qb)=qq(p*n)
 in (p*q,pb+qb))
(Ld pp)/(Ld qq)=Ld (\n->
 let (p,pb)=pp(n*recip q)
     (q,qb)=qq eq
     eq= -(p/(q*q))*n
 in (p/q,pb+qb))

exp (Ld pp)  = Ld (\n->
 let (p,pb)=pp (w*n)
     w=exp p
 in  (w,pb))
sqrt (Ld pp) = Ld (\n->
 let (p,pb)=pp eb
     w=sqrt p; eb=(0.5/w)*n
 in  (w,pb))
```

Pour une fonction quelconque (codée séparément) `f`, si sa ***dérivée formelle*** `f'` est connue (et également codée ; par exemple `f` peut être la fonction sinus hyperbolique, alors la dérivée sera le cosinus hyperbolique), on peut prévoir un « lifting » générique, universel :

```
lift f f' (Ld pp) = Ld (\n->
  let (p,pb)=pp eb
      eb=(f' p)*n
  in (f p,pb))
```

On peut le faire aussi avec des opérateurs binaires.

```
dllift f f1' f2' (Ld pp)(Ld qq)= -- Ops. bin.
  Ld (\n->
    let (p,pb)=pp ep
        (q,qb)=qq eq
        ep=(f1' p q)*n
        eq=(f2' p q)*n
  in (f p q, pb+qb) )
```

Ces constructions ont un goût artificiel et demandent une bonne expérience, mais on peut les enseigner en quelques heures, tandis que les détails de l'approche traditionnelle occupent quelques dizaines de pages, les implémentations impératives sont vraiment longues… Donc, même si l'efficacité de la méthode proposée ici n'est pas idéale, car la mémoire peut être encombrée par des objets différés, sur le plan méthodologique cette « sortie du temps », semble utile.

D'ailleurs, notre programme sera plus efficace, si on ne diffère pas l'évaluation des valeurs « normales », et retarde uniquement les adjoints. Ceci a été implémenté, mais ne sera discuté ici.

Le programme utilisateur a l'air normal, avec des expressions arithmétiques, etc., mais rappelons qu'à la fin on constate qu'il retourne un transformateur d'états, une fonction qui attend *encore* d'être appliquée. On l'applique à 1, à l'adjoint du résultat final : $dz/dz = 1$, et à la fin on récupère $(z, \overline{x} = dz/dx)$.

5 Conclusions

Dans un monde formel, mathématique, la relation de causalité est une relation comme une autre. Dans le modèle fonctionnel de programmation, qui est par excellence statique, le temps n'est qu'un paramètre, alors nous pouvons faire des expériences assez étonnantes avec cette causalité. Le temps qui subit des bifurcations, les références vers les événements futurs, le temps qui va à l'envers, etc., tout ceci devient techniquement modélisable, et le but de ces modèles est simplement de *faciliter la programmation*.

Un chercheur ou ingénieur placé devant un problème logique complexe, surtout dans le domaine de simulation, constate plusieurs embrouillements : des fragments d'un système influencent la totalité, mais le global détermine le comportement des fragments. Le système se comporte téléologiquement, et anticipe ses actions selon un objectif *a priori*, même si ce n'est qu'une façon de décrire ses propriétés.

La possibilité de rompre le lien entre le temps physique et le temps simulé permet d'organiser un programme de manière plus propre. La ***vraie*** causalité est – évidemment – toujours respectée. Nous pourrons programmer la consommation d'un « fil d'Ariane » avant que celui-ci soit placé, mais s'assurant qu'effectivement il sera déroulé comme il faut, plus tard. Ce « plus tard », grâce aux protocoles de la programmation paresseuse devient beaucoup plus contrôlable.

Ces techniques sont difficiles, car l'enseignement des techniques paresseuses de programmation n'est pas populaire en dehors du cursus d'informatique théorique… Cependant, au lieu de décrire en termes généraux la méthodologie exposée, nous avons préféré de donner des exemples *très concrets*, et utiles dans la programmation scientifique. Vu la nécessité d'allouer la mémoire pour les objets différés, les programmes paresseux souffrent d'une certaine inefficacité, qui avec le progrès des techniques d'optimisation deviendra moins importante. Cependant, notre ambition était surtout l'économie du temps humain, non pas celui de la machine.

Nous n'avons pu discuter les réseaux neuronaux ni la cinématique inverse, ce qui constitue actuellement un des sujets de notre recherche dans le domaine d'application des paradigmes fonctionnels. Mais ces sujets démontrent une affinité prononcée avec les problèmes de différentiation automatique. Dans le cas de cinématique ceci est presque évident, il faut construire les inverses des matrices de Jacobi. Dans le cas des réseaux neuronaux, cette affinité a été découverte grâce à l'ingénuosité de Wan et Beaufays [7, 6].

Les langages paresseux dans leur noyau sémantique restent rares. Mais les langages fonctionnels stricts, comme Scheme, ou OCaml [16], grâce à la possibilité de former les *fermetures*, des objets fonctionnels dynamiques, assemblés lors de l'exécution du programme, permettent la construction des séquences différées, ou des chaînes de « promesses » également. Ainsi la portée des techniques paresseuses présentées ici devient beaucoup plus large.

Références

[1] T. Johnsson, *Attribute Grammars as a Functional Programming Paradigm*, Conference on Functional programming Languages and Computer Architecture, Portland, Proceedings : Springer LNCS 274, pp. 154–173, (1987).

[2] C. Welman, *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulations*, Simon Fraser Univ. **(1993)**.

[3] L.B. Rall, *Automatic Differentiation – Techniques and Applications*, Springer Lecture Notes in Comp. Sci., Vol. **120**, (1981).

[4] D. Juedes, *A taxonomy of automatic differentiation tools*. Dans : A. Griewank and G. F. Corliss, editeurs, *Automatic Differentiation of Algorithms : Theory, Implementation, and Application*, SIAM, Philadelphia, Penn., (1991), pp 315–329.

[5] P. Werbos, *Backpropagation Through Time : What It Does and How to Do It*, Proceedings IEEE, special issue on neural networks **2 (78)**, pp. 1550–1560, (1990).

[6] E.A. Wan, F. Beaufays, *Diagrammatic Derivation of gradient Algorithms for Neural Networks*, Neural Computation, **8 :1**,pp. 182–201, (1996), (aussi : autres articles de Eric Wan et Françoise Beaufays).

[7] E.A. Wan, F. Beaufays, *Relating Real-time Backpropagation and Back-propagation Through Time : An Application of Flow Graph Interreciprocity*, Neural Computation **6 :2**, pp. 296–306, (1994)

[8] Site web : **`http://www.haskell.org`** contient les distributions des compilateurs et toute la documentation.

[9] Jerzy Karczmarczuk, *Functional Framework for Sound Synthesis*, PADL '2005, Long Beach, pp. 7–21, (2005).

[10] Kevin Karplus, A. Strong, *Digital Synthesis of Plucked Strings and Drum Timbres*, Computer Mus. Journal **7** :2, (1983), pp. 43–55.

[11] Jerzy Karczmarczuk, *Traitement paresseux et optimisation des suites numériques*, JFLA'2000, Mt St. Michel, pp. 17–30.

[12] P. Wadler, *The Essence of Functional programming*, 19'th Symp. on Princ. of Prog. Lang., Santa Fe, (1992).

[13] Jerzy Karczmarczuk, *Functional Differentiation of Computer Programs*, Higher-Order Symbolic Computations **14**, (2001), pp. 35–57.

[14] R. Giering, T. Kaminski, *Recipes for Adjoint Code Construction*, ACM TOMS, **24(4)**, (1998), pp. 437–474.

[15] J. Karczmarczuk, *Calcul des adjoints et programmation paresseuse*, Journées JFLA '2001, Pontarlier, (2001), pp. 145-156.

[16] Xavier Leroy *et. al.* Module paresseux. Documentation de OCaml, site Web : **`http://caml.inria.fr/pub/docs/manual-ocaml/libref/Lazy.html`**

Structuration d'information spatiale qualitative pour la Recherche d'Information

Julien Lesbegueries[1], Pierre Loustau[1]

Laboratoire d'Informatique LIUPPA,
Université de Pau et des Pays de l'Adour, Avenue de l'Université - BP 576 - 64012 PAU cedex
julien.lesbegueries@univ-pau.fr et pierre.loustau@univ-pau.fr

Résumé

Cet article présente le projet "Pyrénées Itinéraires Virtuels". Ce projet consiste à valoriser un fonds documentaire patrimonial localisé dans le territoire pyrénéen. Dans ce cadre, nous proposons un modèle spatial unifié pour la définition formelle d'entités spatiales. Ce modèle permet de mettre en place un système de recherche d'information basé sur le contenu sémantique de documents multi-formats. L'objectif de ce projet est d'étendre les fonctionnalités de systèmes de gestion de base documentaire classiques en permettant une gestion plus fine des restrictions spatiales dans une recherche. Pour cela nous développons un processus d'Extraction d'Information spécifique basé sur les modèles unifiés. De plus, une réflexion est menée sur l'interprétation numérique des entités spatiales. Un outil de Recherche d'Information utilise alors le traitement sémantique effectué pour retrouver des fragments de documents spatialement pertinents. Un prototype implémentant ce processus est développé afin de valider nos travaux.

Mots-clés : raisonnement spatial, extraction et recherche d'information spatiale

1 Introduction

Les travaux présentés dans cet article sont effectués dans le cadre du projet "Pyrénées Itinéraires Virtuels"[1], dont le but est de valoriser un fonds documentaire patrimonial composé de documents hétérogènes : livres, journaux, cartes postales, lithographies, etc. Ces documents ne sont accessibles que dans les archives des musées ou des bibliothèques. Afi n de permettre un accès plus généralisé, une campagne de numérisation et d'OCRisation[2] a été lancée. Cependant, ce processus n'est pas suffi sant pour mettre en place un système performant de Recherche d'Information, nécessaire à la «re-socialisation» de ce corpus (3).

Ces documents ont la particularité d'être fortement attachés au patrimoine local. Ceci nous a donc amené à concevoir un système de Recherche d'Information spécifi que basé sur les entités géographiques. Des systèmes existants tels que le projet SPIRIT (18) travaillent sur l'indexation spatiale de documents (pages web) et sur l'élaboration de systèmes de recherche d'information effi caces. Nous voulons indexer des entités spatiales plus complexes que celles présentes dans le corpus de SPIRIT. En effet, le type de corpus sur lequel nous travaillons est composé de documents comprenant une multitude d'entités spatiales décrites ou exprimées de manière qualitative. De plus, nous devons tenir compte de l'aspect multi-format d'un corpus historique (texte, image) et créer des modèles assez génériques pour pouvoir être validés sur ces formats.
Pour cela nous avons élaboré un processus d'extraction d'information utilisant la sémantique des documents, plus fi n qu'un processus utilisant les approches statistiques d'indexation. Afi n d'indexer sémantiquement ces documents multi-formats, nous avons bâti pour l'aspect spatial un Modèle Spatial Unifi é (MSU), permettant de fournir une représentation formelle pour ces données non-structurées. Nous nous sommes basés pour cela, sur les travaux du Raisonnement Spatial Qualitatif (RSQ). En effet, la description d'entités spatiales faite dans les textes est cohérente par rapport aux différents aspects du RSQ : topologie, méréologie, orientation, etc. (7). Nous avons défi ni pour chacun de ces aspects, des "relations spatiales" dans le MSU.
Nous avons ensuite mis en place un moteur d'interprétation numérique de la représentation venant du MSU afi n d'utiliser les entités spatiales extraites dans un système de Recherche d'Information spatial.

L'article se compose de deux parties. La première s'intéresse à l'extraction d'information spatiale, nous donnons un aperçu des travaux qui sont utilisés pour la proposition d'un Modèle Spatial Unifi é. Puis une deuxième partie présente une utilisation de ce modèle dans le cadre d'un système de recherche spatiale d'information.

[1]Le projet PIV est soutenu par la Communauté d'Agglomération de Pau et la Médiathèque Intercommunale à Dimension Régionale de l'Agglomération Paloise.

[2]Reconnaissance Optique de Caractères : permet de récupérer du texte à partir d'une image.

2 Extraction de la sémantique spatiale des documents

L'information géographique contenue dans notre corpus est présente sous plusieurs modes d'expression. Chaque mode a ses spécifi cités. Si le texte est effi cace pour décrire l'expression d'un phénomène sur un lieu géographique, une carte est plus appropriée lorsque l'on évoque l'organisation spatiale complexe d'un phénomène. Cependant, quel que soit le mode d'expression, cette information géographique apparaît sous forme d'Entités Géographiques (EGs). Ces dernières se composent d'une Entité Spatiale (ES), d'une Entité Temporelle (ET) qui peut être implicite et d'un phénomène (17). Afi n de traiter correctement l'information géographique, une analyse fi ne des aspects spatiaux et temporels est obligatoire. Nous nous appuyons ici sur une approche sémantique qui a déjà fait ses preuves (5; 20).
Nous nous focaliserons dans cet article uniquement sur l'aspect spatial des EGs.

Le concept "cible/site" : Les travaux des linguistes montrent la manière particulière qu'a l'humain de se représenter une information spatiale lorsqu'elle est évoquée dans le langage écrit (1). Faire référence à un lieu met en jeu plusieurs éléments et ces éléments respectent une position dans la phrase. (19) propose le concept de *cible/site* : dans le langage écrit, la cible correspond à l'objet de la description, le site à la référence. Notre hypothèse est d'étendre ce concept à d'autres modes d'expression comme l'image par exemple.

Modèles unifiés pour le RSQ : Afi n de développer une Recherche d'Information effi cace, nous proposons des modèles unifi és pour représenter le temps et l'espace. Contrairement à (9), (11) ou à GML[3] qui gère l'information spatiale du point de vue des bases de données, les modèles que nous proposons doivent supporter l'information géographique contenue dans notre corpus. Celle-ci est non structurée, polysémique, et dépendante du contexte.

A partir des travaux des linguistes, nous défi nissons une entité spatiale récursivement grâce à d'autres entités spatiales et à des relations spatiales (Figure 1). Prenons l'exemple de l'expression *"le nord de la ligne Pau-Biarritz"*. Dans cette expression l'entité spatiale évoquée est (i) tout d'abord défi nie par deux sites (ici deux entités nommées : *Biarritz* et *Pau*), (ii) puis le terme *ligne* exprime une relation géométrique linéaire entre ces deux sites. Enfi n, (iii) *"le nord de"* exprime une orientation pour mettre en évidence l'espace évoqué.
En résumé, une entité spatiale peut-être (fi gure 1) :

- une Entité Spatiale Absolue (ES_A) lorsqu'il s'agit d'une référence directe à un espace géo-localisable (une entité nommée par exemple),

[3]GML : Geography Markup Language (http://opengis.net/gml)

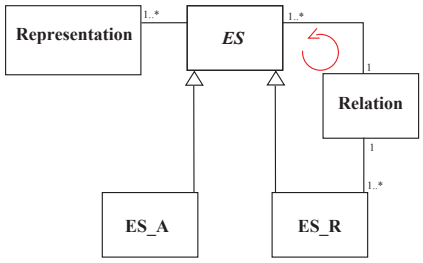


FIG. 1 – Schéma UML simplifi é du Modèle Spatial Unifi é

- une Entité Spatiale Relative (ES_R) si elle est défi nie à l'aide d'au moins une autre entité spatiale et de relations spatiales.

Cette approche concorde avec la façon de défi nir un formalisme spatial dans le domaine du RSQ. En effet elle consiste dans un premier temps à défi nir des entités spatiales basiques, puis à défi nir des relations atomiques qualitatives susceptibles de relier ces entités spatiales entre elles (4).

Dans notre cas les entités basiques sont les toponymes que l'on peut rencontrer dans une base documentaire (noms de villes cités dans des textes, noms de vallées dessinées dans des lithographies, etc.) et les relations sont des relations d'ordre topologique, d'orientation et de distance. Nous avons défi ni cinq relations spatiales qui sont l'adjacence, l'orientation, la distance, l'inclusion (6; 7) et la relation géométrique. Cette dernière correspond à une relation entre au moins deux entités et construisant dans le sens commun une fi gure géométrique (exemple : «le triangle Pau Bordeaux Toulouse», «l'axe Pau Bayonne», etc.).

Cette défi nition récursive peut aussi s'appliquer à d'autres modes d'expression. Dans une image, les entités nommées peuvent être représentées par des symboles ponctuels alors que des entités spatiales plus complexes peuvent s'évoquer par des relations entre ces entités spatiales nommées : une ligne puis deux zones contrastées par exemple.

Analyse de la sémantique : Le processus d'Extraction de l'Information que nous préconisons ici se compose de quatre étapes principales : la lemmatisation[4], l'analyse lexicale et morphologique, l'analyse syntaxique et l'analyse sémantique (12). Ce processus d'extraction d'information a été validé pour les documents textuels par le développement de chaînes de traitement linguistique utilisant la plate-forme Linguastream[5] et des grammaires écrites en Prolog. Il produit des instances de Modèle Spatial Unifi é (MSU) à chaque fois qu'une entité spatiale est détectée dans notre corpus.
Concernant les documents images, la validation des modèles a été faite manuellement mais il reste à automatiser la détection d'entités spatiales.

[4]fonction qui, à un mot, associe sa forme canonique.
[5]Linguastream : http://www.linguastream.org

3 Structuration et utilisation d'entités décrites qualitativement dans un système de Recherche d'Information

Le MSU défi ni dans la section précédente permet de garantir une homogénéité dans la description des entités spatiales, quel que soit le mode d'expression employé. En effet, nous parlons généralement de description et de raisonnement qualitatif de l'espace pour le mode d'expression textuel. Mais il existe aussi des travaux sur la description qualitative d'entités spatiales pour d'autres modes d'expression, comme dans une image par exemple (2). Ce modèle unifi é consiste donc en une première étape vers une représentation formelle de ces entités. Cependant, cette représentation générique n'est pas adaptée pour être utilisée directement dans un système de recherche d'information. Il est donc nécessaire de la traduire en une représentation plus propice à la comparaison et au classement, telle une représentation numérique et géo-référencée.
Le problème est alors de défi nir une méthode de traduction du MSU vers une telle représentation, dépendante du contexte et du degré de précision requis par le système de recherche d'information. Cette dernière représentation peut alors être utilisée comme comparateur et permet de calculer un degré de pertinence pour les entités lors du traitement d'une requête spatiale. Il est judicieux, pour ce calcul de représentation, d'utiliser les fonctionnalités des Systèmes d'Information Géographique. En effet ceux-ci se comportent comme des systèmes de gestion de bases de données classiques (adaptés pour l'indexation) et comportent en plus des fonctionnalités permettant le calcul de zones géo-référencées.

D'autre part, utiliser une ontologie pour fabriquer des représentations compréhensibles à la fois par l'homme et la machine est une idée assez répandue, notamment dans le domaine du spatial (16). (14) propose un modèle d'interprétation automatique de carte géographique, reposant sur un modèle de traduction basé sur une ontologie qui décrit le domaine d'étude (des objets géographiques pouvant être présents sur une carte). L'approche retenue dans notre cas est de pouvoir utiliser n'importe quelle ressource géographique (telle qu'une ontologie spatiale) pour fabriquer des interprétations spatiales pouvant être utilisées comme comparateurs dans un système de recherche d'information.

3.1 Problématique d'interprétations multiples :

Il est important de considérer qu'une entité spatiale peut avoir de multiples représentations. Par exemple une ville défi nie par une zone géométrique dans un Système d'Information Géographique (SIG) peut être représentée par un point, un polygone, une boîte englobante, etc. La représentation est choisie pour un cas précis selon la base de connaissances employée ou selon les besoins. C'est pourquoi il est préférable de laisser la possibilité d'utiliser une base de connaissances moins riche ou simplifi ée.
Il est possible d'utiliser les SIG pour la phase d'interprétation. Nous pouvons aussi employer d'autres bases de connaissances tel un thésaurus décrivant un ensemble d'entités géographiques mais de manière non numérique. En effet, dans certains cas, l'utilisation de données quantitatives n'est pas adaptée (7; 15).

Nous avons donc défi ni et implémenté un Modèle Spatial Unifi é (MSU) qui se place en amont dans la phase de structuration de l'information spatiale, assez générique pour pouvoir être utilisé avec les différentes bases de connaissances éventuelles. La fi gure 2 est un schéma montrant les exploitations possibles d'une entité spatiale extraite. Il lui est associé une instance du MSU qui servira à produire une indexation appropriée en fonction de la ressource choisie.

3.2 Une couche SIG pour ontologie :

La communauté des SIG s'intéresse de plus en plus aux ontologies décrivant les phénomènes spatiaux (13). L'association d'un SIG avec des ontologies décrivant les domaines du spatial est une pratique courante. (10) utilise les ontologies dans un système de gestion de données géographiques inter-opérables. Elles servent dans ce cas à assister la recherche d'information géographique et à faire inter-opérer plusieurs SIG ensemble.

Nous proposons ici de considérer un SIG et des couches de données géographiques comme base de connaissances servant à créer une représentation numérique des entités spatiales extraites de notre corpus. En effet, nous nous servons des concepts défi nis dans les couches de données d'un SIG, leur structure relationnelle et les fonctionnalités propres aux SIG pour interpréter nos entités décrites grâce au modèle unifi é. Il en ressort pour chacune d'entre elles une zone géo-référencée manipulable par un moteur de recherche spatial. La fi gure 2 présente le fonctionnement d'un tel système.
Dans le cas de l'utilisation d'un SIG, nous pouvons imaginer un format de sortie en GML pour l'interprétation numérique. D'autres formats peuvent aussi être envisagés, en XML, etc.

3.3 Exemple d'utilisation d'un SIG :

Dans le cadre du projet PIV a été développé un prototype de système d'extraction et de recherche d'information spatiale (12). Dans ce prototype, le moteur d'interprétation utilise un SIG contenant les communes de France comme base de connaissances. Prenons l'exemple d'une interprétation numérique faite à partir de notre modèle. Dans un document se trouve l'entité *"à l'est d'Eaux-Bonnes"*. Celle-ci est extraite grâce à un traitement sémantique, puis une

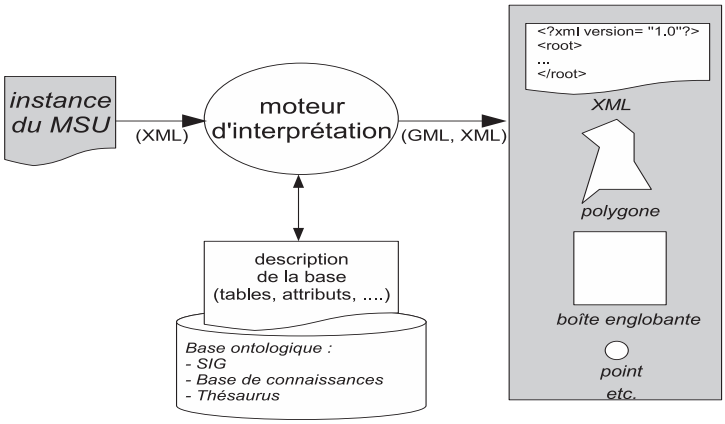


FIG. 2 – Méthode d'interprétation multiple - du MSU à l'indexation

```
<entite_geographique id="4" id_paragraphe="2">
  <eg_r>
    <libelle>est d'Eaux-Bonnes</libelle>
      <relation>
        <orientation><type>est</type></orientation>
      </relation>
    <eg_a>
      <libelle>Eaux-Bonnes</libelle>
       <type>village</type>
    </eg_a>
  </eg_r>
  <representation>
    <boite_englobante>
      <x_min>360689.2</x_min><y_min>1752718.6</y_min>
      <x_max>389050.6</x_max><y_max>1789151.3</y_max>
    </boite_englobante>
  </representation>
</entite_geographique>
```

FIG. 3 – Instance du MSU (en grisé l'ajout d'une représentation)

instance du MSU est créée. Ensuite le moteur d'interprétation, à l'aide d'une couche SIG, propose en sortie une figure géométrique géo-référencée (ici une boîte englobante) représentant l'est d'Eaux-Bonnes (figure 3).
Le moteur d'interprétation consiste en un ensemble de fonctions SQL avancées[6] définies à l'intérieur d'un SIG, construisant des zones géo-référencées à partir d'instances de MSU. Nous définissons cette interprétation de manière plus ou moins subjective, en fonction de l'aspect plus ou moins qualitatif des entités décrites. A titre d'exemple, nous pouvons citer les fonctions «GeomUnion(), Buffer(), Translate(), Scale(), etc.»[7] qui peuvent être utilisées pour définir des fonctions du type «PresDe(), AuSudDe(), DistanceDe(), Entre(), etc.». Comme on le voit dans la Figure 2, le moteur d'interprétation peut alors produire une zone géo-référencée directement stockable dans un SIG. Nous construisons alors un index «spatial» en calculant cette zone pour chaque entité spatiale extraite d'un corpus. Pour l'exemple *"à l'est d'Eaux-Bonnes"* la couche SIG fournit la boîte englobante de la commune *"Eaux-Bonnes"*, puis un algorithme effectue une translation de cette boîte en fonction de la relation *"à l'est de"*. De la même manière, à chaque relation définie dans le modèle spatial unifié est associée une transformation adéquate (homothétie, translation et composition de ces deux transformations).

Cet index est ensuite utilisé lors de la recherche d'information, permettant le calcul d'un degré de pertinence spatial entre les zones géo-référencées indexées pour la base documentaire et les zones de la requête. Ce degré de pertinence peut aussi se faire à l'aide de fonctionnalités SIG de calcul de surface et d'intersection. Prenons l'exemple de la requête suivante.

```
SELECT nomcommune, Area(Intersection(the-
    geom, geom-requete)) AS recouvrement
FROM communes
WHERE departement='64'
ORDER BY recouvrement DESC;
```

Elle retourne le nom des communes (nomcommune) des Pyrénées-Atlantiques (64) et leur surface de recouvrement avec la géométrie de la requête (geom-requete), de la plus grande surface à la plus petite. Nous pouvons alors retourner les résultats spatialement les plus pertinents, en considérant que plus la surface d'intersection est grande, plus le degré de pertinence est élevé.

3.4 Expansion de requêtes spatiales et glissement de sens

La recherche d'information basée sur des critères spatiaux implique la gestion de requêtes spatiales. La première idée était de se servir du processus existant pour l'extraction et l'indexation d'entités spatiales et de construire ces

[6] c'est-à-dire des fonctions SQL utilisant les fonctionnalités supplémentaire propres aux SIG.
[7] http://postgis.refractions.net/docs/ch06.html

requêtes en texte libre. Cependant nous pouvons imaginer plusieurs améliorations :

- d'une part, une étape de vérification visuelle, permettant de retourner à l'utilisateur l'interprétation de sa requête faite par la machine sur un fond cartographique. Un prototype utilisant les services web «ViaMichelin» de génération de carte[8] a été créé pour valider les zones géo-référencées calculées par PIV.
- D'autre part, nous proposons la conception d'outils d'expansion (ou de réduction) de requêtes par le biais de glissements de sens effectués sur la requête (8).

Ces améliorations impliquent la création de modèles pour les glissements de sens, capables de modifier l'interprétation originale des entités spatiales. Pour cela, une réflexion sur les proximités entre les différentes relations définies dans le MSU doit être menée.

4 Conclusion

Nous présentons dans cet article les premiers travaux d'exploitation d'un corpus territorialisé à travers sa dimension spatiale. Un travail sur le raisonnement spatial qualitatif a permis de concevoir une méthode de structuration de l'information géographique contenue dans les documents. Cette méthode a été validée par l'implémentation d'un prototype de recherche d'information spatial.
Nous avons aussi mis en avant la nécessité de concevoir un système générique d'interprétation afin de pouvoir mettre en œuvre des algorithmes de comparaison spatiale. Cette interprétation doit pouvoir utiliser différentes ressources (ontologiques, géographiques, etc.) afin de s'adapter à différents processus de recherche d'information. La solution présentée dans cet article est une solution qui utilise un SIG, mais nous pourrions très bien définir d'autres interprétations utilisant des ressources non numériques (ontologie, thésaurus, etc.).

La prochaine perspective est d'ajouter à ce genre de systèmes une dimension temporelle et thématique afin d'indexer des entités géographiques complètes (comprenant la dimension spatiale, temporelle et un phénomène).

Références

[1] A. Borillo. *L'espace et son expression en français*. L'essentiel. Ophrys, 1998.

[2] Gilberto Camara, Max J. Egenhofer, Frederico T. Fonseca et Antonio Miguel Vieira Monteiro. What's in an image ? In *Spatial Information Theory*, pages 474–488, 2001.

[3] Jean Casenave, Christophe Marquesuzaà, Pantchika Dagorret et Mauro Gaio. La revitalisation numérique du patrimoine littéraire territorialisé. In *EBSI-ENSSIB, Montréal*. 2004.

[4] Khalil Challita. *Problèmes de satisfaction de contraintes spatiales : de l'algèbre des régions à la géométrie affine*. PhD thesis, Université de Toulouse III, 2005.

[5] T Charnois, Y Mathet, P Enjalbert et F Bilhaut. Geographic reference analysis for geographic document querying. Workshop of the NAACL-HLT Conference, Association for Computational Linguistic, 2003.

[6] A G Cohn et S M Hazarika. Qualitative spatial representation and reasoning : An overview. *Fundamenta Informaticae*, 46(1-2) :1–29, 2001.

[7] Anthony G. Cohn. Qualitative spatial representation and reasoning techniques. In *KI '97 : Proceedings of the 21st Annual German Conference on Artificial Intelligence*, pages 1–30, London, UK, 1997. Springer-Verlag.

[8] F Dumoncel, M Ould Ahmed Limam et M Gaio. Feedback sémantique dans le cadre d'un environnement de recherche d'informations géographiques. In *15eme conférence francophone sur l'Interaction Homme-Machine (IHM'03), Caen, France*, novembre 2003.

[9] Max J. Egenhofer. Toward the semantic geospatial web. In *GIS '02 : Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, pages 1–4. ACM Press, 2002.

[10] Frederico T. Fonseca et Max J. Egenhofer. Ontology-driven geographic information systems. In *GIS '99 : Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, pages 14–19, New York, NY, USA, 1999. ACM Press.

[11] Linda L. Hill. Core elements of digital gazetteers : Placenames, categories, and footprints. In *ECDL '00 : Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, pages 280–290. Springer-Verlag, 2000.

[12] Julien Lesbegueries, Mauro Gaio, Pierre Loustau et Christian Sallaberry. Geographical information access for non-structured data. ACM SAC ASIIS - Dijon France, 23-27 avril, 2006.

[13] David Mark, Max Egenhofer, Stephen Hirtle et Barry Smith. Ontological foundations for geographic information science, 2001.

[14] Victor Montes de Oca Morales. Template-based geospatial knowledge representation. Short Paper Geos2005, 2005.

[8] http://ws.viamichelin.com/wswebsite/fra/jsp/hme/MaHomePage.jsp

[15] Philippe Muller. *Éléments d'une théorie du mouvement pour la modélisation du raisonnement spatio-temporel de sens commun*. PhD thesis, Université Paul Sabatier, 1998.

[16] Edyta Przytula-Machrouh, Gérard Ligozat et Michel Denis. Vers des ontologies transmodales pour la description d'itinéraires. *Revue internationale de Géomatique - European Journal of GIS and Spatial Analysis*, 2004.

[17] E. Lynn Usery. Multidimensional representation of geographic features. Technical report, U.S Geological Survey (USGS), 2003.

[18] S Vaid, C.B. Jones, H Joho et M Sanderson. Spatio-textual indexing for geographical search on the web. In *à paraître : Proceedings of 9th International Symposium on Spatial and Temporal Databases*, 2005.

[19] Claude Vandeloise. *L'espace en français*. 1986.

[20] A Widlöcher, E. Faurot et F. Bilhaut. Multimodal indexation of contrastive structures in geographical documents. In *Actes RIAO 2004, Avignon*, pages p. 555–570, 2004.