Généralisation de graphes conceptuels

Gaël De Chalendar, Brigitte Grau, Olivier Ferret

Généralisation de graphes conceptuels

Conceptual graphs generalization

Gaël de Chalendar, Brigitte Grau et Olivier Ferret

LIMSI-CNRS

BP 133.91403 Orsay, France
mél : (gael, grau, ferret)@limsi.fr

Résumé : *Dans l'optique de former des descriptions prototypiques de situations concrètes, nous avons déjà réalisé un système, nommé MLK (Memorization for Learning Knowledge), permettant de regrouper des événements relatifs à des situations similaires à partir de descriptions trouvées dans des textes, événements représentés par des graphes conceptuels. L'une des étapes pour construire ces prototypes consiste à généraliser certains graphes similaires afin de produire la description cherchée. Dans cet article, nous présentons un algorithme de généralisation de graphes conceptuels procédant par regroupement ascendant, borné par l'utilisation de coûts sur les opérations de généralisation afin de limiter l'espace de recherche.*

Mots Clés : traitement automatique de la langue, apprentissage non supervisé, généralisation de graphes conceptuels

Summary: *In order to build descriptions of prototypical situations, we first developed a system, MLK (Memorization for Learning Knowledge), allowing to gather events related to similar situations starting from descriptions found in texts, these events being represented by conceptual graphs. One of the stages to build these prototypes consists in generalizing some similar graphs in order to produce a description. In this paper, we present a cost bounded algorithm of conceptual graphs generalization, proceeding by ascending clustering. The use of costs on the operations of generalization allows to control the growth of the search space.*

Keywords: natural language processing, unsupervised learning, conceptual graphs generalization.

1. Introduction

Un système de compréhension de textes utilise, entre autres types de savoirs, la connaissance des situations concrètes développées dans un texte. Une telle connaissance est souvent représentée par des schémas dont le contenu décrit les personnages et les événements, par ordre chronologique. Cependant, la définition manuelle de ces schémas est une tâche difficile à réaliser, même sur un domaine limité. Et si l'acquisition automatique de telles connaissances a été étudiée [Lebowitz, 1983], [Mooney et DeJong, 1985], [Pazzani, 1988] et [Ram, 1993], ces systèmes visent à apprendre des situations qui spécialisent des situations générales prédéfinies. Dans le cadre d'une approche ne formant pas d'hypothèses sur l'existence de telles connaissances, nous avons conçu le système MLK, [Ferret et Grau, 1997 et 1998], capable d'apprendre à partir de l'accumulation de sa propre expérience. MLK mémorise chaque situation spécifique trouvée dans des textes en l'agrégeant avec une autre déjà existante si une similitude entre leurs événements est reconnue. Ce processus mène à construire incrémentalement des unités thématiques agrégées (UT), qui sont les précurseurs de schémas généraux. Chaque événement à l'intérieur d'une UT est représenté par un graphe conceptuel [Sowa, 1984] muni d'un poids. Afin de pouvoir élaborer une description générale d'une situation à partir de ces UTs, nous avons été amenés à étudier la généralisation d'événements, donc de graphes conceptuels, afin de trouver un niveau de description rendant compte des différentes formulations d'un même type d'événement. Ce type de problème s'inscrit dans le cadre d'un processus de regroupement ascendant à partir d'exemples positifs. Des processus de généralisation de graphes conceptuels ont déjà été proposés [Mineau, 1992], [Bournaud, 1996] mais ils ne généralisent que les concepts, et non la structure des graphes elle-même. Une généralisation à la fois des concepts et des graphes eux-mêmes entraîne une explosion combinatoire lors du calcul des généralisations possibles. Par conséquent, nous avons développé un algorithme de généralisation borné par des coûts associés aux opérateurs de généralisation afin de limiter l'espace de recherche, algorithme exploitant les connaissances sémantiques du domaine pour produire des généralisations informatives.

2. Vue d'ensemble du système

Le système dispose en entrée d'un ensemble de graphes conceptuels décrivant des événements relatifs à un même type de situation, par exemple *entrer dans une maison*, *attaquer une femme*, *poignarder un homme avec un couteau*, *arrêter le meurtrier*, etc. . Afin d'élaborer une description de la situation à partir

de ces graphes, l'une des étapes est de trouver un niveau de description regroupant, parmi ces événements, ceux qui sont relatifs à un même type d'événement. En reprenant l'exemple, cela consisterait à former une description généralisant *l'attaque d'une femme* et le fait de *poignarder un homme*, alors que *entrer* et *arrêter* resteraient tels quels. Le problème peut donc être posé ainsi : étant donné un ensemble de graphes, trouver parmi ceux-ci ceux qui se généralisent, tout en conservant un niveau de description informatif. En effet, il ne s'agit pas de trouver la plus grande généralisation commune, mais un niveau de description gardant la spécificité des types d'événement représentés par les graphes. Toujours en reprenant l'exemple, le but n'est pas de généraliser tous les graphes pour former un seul graphe qui serait *faire une action*.

La méthode que nous proposons consiste à former un sous-ensemble des généralisations possibles à partir de chaque graphe, en généralisant les concepts ainsi qu'en supprimant des relations. Les opérations de généralisation sont effectuées en faisant référence aux connaissances sémantiques du domaine : l'ontologie des concepts et leur description éventuelle en terme de contraintes sur leurs arguments. Cela évite la construction de sur-généralisations qui n'auraient plus de sens. Produire toutes les généralisations possibles de chaque graphe n'est pas concevable, même en utilisant les connaissances du domaine pour limiter les généralisations. Par conséquent, pour limiter la taille de l'espace de généralisation, un coût est associé aux opérateurs de généralisation, ce qui permet de calculer un coût pour chaque graphe généralisé et de limiter leur nombre en se référant à un seuil donné. Ces coûts sont définis en fonction de la tâche, et rendent compte du niveau de généralisation que l'on cherche à produire. Les généralisations produites sont organisées en configurations, permettant de savoir quels graphes de départ sont généralisés et ceux qui ne le sont pas. Ces configurations représentent les différentes possibilités pour décrire la situation initiale. Le choix du niveau de description est alors effectué, en fonction du nombre de graphes généralisés et du coût associé à chaque configuration. Ainsi, le niveau de description que l'on forme est contraint à la fois par les connaissances du domaine et par les coûts associés aux graphes généralisés. Cette technique nous permet de contourner la difficulté posée par l'impossibilité dans notre cadre de définir des exemples négatifs, ce qui nous empêche d'utiliser un grand nombre de méthodes d'apprentissage.

3. Graphes conceptuels

Les graphes conceptuels sont des graphes bipartites, ayant pour nœuds des concepts et des relations. Un concept est une instance d'un type appartenant à un treillis. De ce fait tout couple de types de concept possède un sur-type commun minimal et un sous-type commun maximal. Les connaissances sémantiques du domaine sont définies par ce treillis et par les graphes canoniques associés à certains types afin de préciser les rôles casuels qui peuvent leur être associés, comme 'agent', 'objet', etc., ainsi que leurs contraintes sémantiques, c'est-à-dire les types de concept pouvant remplir ces rôles (i.e. relations) (cf. Figure 1). Les types de relation sont aussi organisés en treillis

[Poignarder]-
(agent) → [Humain]
(dest) → [Corps]
(instrument) → [Arme-Blanche]

Figure 1 : Un graphe canonique

Les graphes conceptuels représentant les événements (cf. Figure 2) sont obtenus en appliquant des règles de formation à un ou plusieurs graphes canoniques, ces règles respectant la canonicité des graphes, c'est-à-dire que les graphes qui en résultent respectent toujours les contraintes définies dans la base. Parmi ces règles, l'opération de jointure maximale groupe deux graphes sur la base des couples de concepts communs ou plus exactement de concepts possédant un sous-type commun maximal, et la restriction de type remplace le type d'un concept par un sous-type. Un graphe résultant de l'application de ces opérations est une spécialisation d'un (ou plusieurs) graphe(s) initial(aux). Ainsi le graphe de la Figure 2 a été formé en restreignant les types des graphes canoniques de *Poignarder* et de *Corps* et en faisant une jointure de ces deux graphes. On se reportera à [Ellis, 1992] pour une présentation détaillée des règles de formation et de la subsomption dans les graphes conceptuels.

[Poignarder]-
(agent) → [Homme]
(dest) → [Corps] → (partieDe) → [Homme]
(instrument) → [Couteau]

Figure 2 : Un événement sous forme de graphe conceptuel

Nos graphes représentant des événements, nous distinguons le concept représentatif du type d'événement, appelé prédicat (Poignarder dans la Figure 2). En effet ce type de concept joue un rôle privilégié dans le cadre de notre application.

Les types de concept autres que le prédicat résultent d'une abstraction ayant eu lieu lors de la construction des situations dans MLK. Ils généralisent déjà les instances trouvées dans les textes de départ. On pourra se reporter à [Ferret et Grau, 1997 et 1998] pour trouver les détails du processus de formation de ces graphes.

4. Généralisation de graphes

4.1. Principes

Certains graphes appartenant à l'ensemble à généraliser peuvent contenir des prédicats différents mais être très semblables, comme dans l'exemple de la figure 3. Si deux graphes sont identiques, à l'exception de leur prédicat, et si ces derniers sont sémantiquement proches, les graphes devront être remplacés par un nouveau graphe qui les généralisera.

Le même principe s'applique s'ils diffèrent seulement par quelques détails dans les arguments des prédicats. Le problème est alors de trouver la plus petite généralisation commune de plusieurs graphes, sachant que l'on ne connaît pas *a priori* les graphes de l'ensemble de départ pouvant être généralisés.

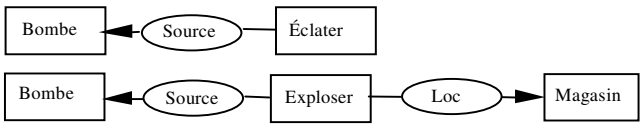

Figure 3 : Deux graphes conceptuels semblables avec des structures et des concepts différents

Un tel problème est similaire à celui du regroupement conceptuel, avec les graphes conceptuels comme langage de description des concepts à apprendre [Michalski et Stepp, 1993]. Les graphes conceptuels initiaux sont équivalents à des formules logiques du premier ordre, formées d'une conjonction de prédicats positifs. Notre problème est proche du travail de Mineau et de Bournaud. Même si nous ne recherchons pas une classification, ce qui constitue le problème qu'ils cherchent à résoudre, l'espace de généralisation doit aussi être calculé afin de choisir le niveau de généralisation que nous souhaitons atteindre. Dans ces travaux, les graphes sont représentés par un ensemble de relations pour éviter des comparaisons de graphes, qui est un problème NP-complet. Cependant, cette approche ne s'applique pas dans notre cas car nous modifions la structure des graphes en supprimant des relations. Nous devons donc raisonner sur les graphes eux-mêmes, afin de maintenir leur connexité en cas de suppression de relation. Le problème est alors de contrôler la taille de l'espace de généralisation tout en produisant des généralisations pertinentes, c'est-à-dire en évitant les sur-généralisations. À cette fin, nous proposons un algorithme borné par des coûts associés aux opérateurs et qui utilise les connaissances sémantiques du domaine. Cet algorithme a été conçu en faisant abstraction de certaines spécificités de notre application, de manière à proposer un processus général de généralisation de graphes conceptuels. Seul le repérage d'un concept privilégié dans un graphe, le prédicat, est ici spécifique, mais cette notion est une caractéristique pouvant appartenir à d'autres types d'application.

Nous avons défini trois opérateurs primitifs de généralisation, un coût étant associé à chacun d'eux :

1. généralisation de concept : substitution d'un concept dans le graphe par son sur-type,
2. généralisation de relation : substitution d'une relation dans le graphe par son sur-type,
3. suppression de relation : suppression d'une relation dans le graphe.

Nous n'avons pas défini d'opérateur de suppression de concept car la suppression d'un concept entraîne la suppression de toutes les relations qui lui sont liées. Si l'on reprend l'exemple de la Figure 3, la suppression du concept *Magasin* entraîne la suppression de la relation *Loc*, et inversement la suppression de la relation *Loc* entraîne la suppression du concept *Magasin*, de manière à conserver un graphe connexe (dans ce cas nous conservons la composante connexe du graphe contenant le prédicat). La seule suppression des relations est donc suffisante puisque les résultats de la suppression de concept sont strictement inclus dans les résultats de la suppression de relation, celle-ci pouvant entraîner la suppression de plusieurs concepts et relations. Ces opérateurs induisent un ordre partiel entre les graphes résultants tel qu'il est défini dans [Sowa, 1984] et [Ellis, 1992]. Plus précisément, la relation de subsomption définie pour les graphes est la relation induite par l'existence d'une projection entre deux graphes [Champesme, 1995].

Nous associons un coût à chaque opération car nous n'avons ni une théorie du domaine (des connaissances générales sur les situations à produire par exemple), ni une caractérisation de la connaissance que l'on veut apprendre qui pourrait nous fournir une preuve formelle de la qualité de la généralisation. Nous utilisons une évaluation empirique fondée sur les connaissances sémantiques et une estimation du coût de la perte d'information associée aux généralisations.

Afin d'associer un coût à un opérateur, nous avons raisonné sur l'effet de chaque opération de généralisation sur les graphes en terme de perte d'information. Ainsi, dans notre contexte, le coût de chaque opération est relatif au concept modifié : prédicat ou non, partie du graphe canonique du prédicat ou non. En effet, lorsqu'on généralise un concept n'appartenant pas au graphe canonique du prédicat (nommé concept quelconque), la généralisation s'effectue sur des caractéristiques plus périphériques de l'événement. La suppression de relation, qui rappelons-le cause la suppression de concepts, entraîne la suppression de ces caractéristiques. Notons qu'on ne peut supprimer des relations appartenant au graphe canonique sous peine de perte du sens de l'information exprimé dans le graphe (la canonicité du graphe résultant n'est plus vérifiée). Enfin, l'opération que nous considérons la plus coûteuse est la généralisation du prédicat, car, comme nous l'avons vu lors de la présentation générale, nous voulons éviter de trop généraliser les prédicats afin de conserver des types d'événements restant spécifiques par rapport à la situation. De ce fait, la plus petite généralisation possible est le graphe dont les concepts autres que le prédicat sont généralisés de manière préférentielle.

Quelques points sont à noter. D'abord, il n'est pas possible de supprimer le prédicat puisque le graphe résultant (ou les graphes) n'aurait plus de sens quant à l'événement décrit. En second lieu, si deux ou plusieurs graphes non-connexes existent après une suppression, nous ne conservons que le graphe contenant le prédicat. Ensuite, un concept n'est jamais généralisé en un concept plus général que celui qui lui correspond dans le graphe canonique de son prédicat, sinon la canonicité serait perdue. Enfin, en généralisant le prédicat, une vérification est effectuée pour s'assurer de la canonicité du graphe résultant par rapport au graphe canonique associé au nouveau type de concept du prédicat.

Nous avons donc défini la relation d'ordre suivante sur les coûts des opérateurs de généralisation, cet ordre caractérisant la perte d'information :

> généralisation d'un concept quelconque P généralisation d'un concept du graphe canonique P suppression d'une relation P généralisation du prédicat.

Plus que les valeurs définissant les coûts en absolu, c'est l'ordre relatif de ces coûts qui est important. L'attribution d'une valeur aux différentes opérations est liée, quant à elle, à la valeur limite que l'on fixe pour calculer une généralisation, c'est-à-dire au nombre total d'opérations qu'il est vraisemblable de réaliser et à l'importance relative d'une opération par rapport aux autres. Ces valeurs sont à fixer selon l'application et donc à déterminer de façon expérimentale.

4.2. L'algorithme de généralisation

Une étape de l'algorithme consiste à construire l'espace de généralisation contenant au départ les graphes conceptuels à généraliser, les nœuds racines. Lors du calcul, chaque nœud produit à partir de l'un des nœuds déjà présent dans l'espace de généralisation est inséré dans cet espace s'il n'existe pas déjà. Un nœud de profondeur *n* est le résultat de *n* opérations de généralisation sur un graphe conceptuel racine. Par exemple, si un graphe conceptuel possède trois concepts et deux relations, on peut lui appliquer cinq généralisations de concept ou de relation et deux opérations de suppression de relation qui produiront, au plus, sept nouveaux nœuds différents. Le coût associé à chaque nœud, relatif au chemin jusqu'au nœud racine, est égal au coût associé à son père plus le coût associé à l'opération qui lui a donné naissance. Le coût pour une racine est zéro. Si cet algorithme est appliqué jusqu'à ce qu'une généralisation commune de tous les graphes soit trouvée, si elle existe, il est exponentiel dans le cas général. Borner le calcul des généralisations par un seuil sur leur coût le ramène à une complexité pratique raisonnable (ce point sera détaillé dans la section suivante) et toutes les généralisations inférieures à ce seuil sont calculées.
La figure 4 montre un exemple dans lequel trois graphes conceptuels (les nœuds racines x1, x2, x3) ont subi le processus de généralisation qui a permis d'engendrer respectivement les graphes (a, b, c, d), (c, b, e) et (e, f, g, d). La valeur associée aux arcs correspond au coût de l'opération appliquée.

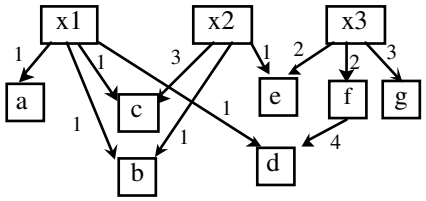

Figure 4 : Exemple d'un espace de généralisation pour 3 graphes agrégés

Soit *E* l'ensemble des graphes de départ, E = {gi} et n = |E|.
Soit *Cgi* les ensembles de graphes généralisés à partir de chaque graphe racine *gi*.

Soit *G*, l'espace de généralisation, l'ensemble de tous les graphes généralisés déjà produits. A chaque graphe généralisé est associée la liste des graphes racines dont il est une généralisation.
Soit *Cp* l'ensemble des configurations partielles déjà produites, initialisé à vide.
Soit *cgp* le coût de généralisation d'un prédicat
Soit *la*, le seuil d'arrêt de la généralisation
Notation : a -> b : a associé à b

Initialisation
Pour tout $g \in E$, et **Pour tout** $g' \in E$ **faire**
  Créer un ensemble *Cg*
  si le prédicat de *g* possède un sur-type commun *sc* avec le prédicat de *g'* situé à une distance *d*, telle que *d*cgp<la*
  alors,
    créer une copie g_c de *g*,
    remplacer le prédicat de g_c par *sc*
    ajouter g_c à *Cg*
    ajouter *g* à *G*
  finSi
finFaire

Abstraction
Tant que il existe un nœud à développer **faire**
  Pour tout $Ci \in C$, **faire**
    Sélectionner un nœud *nd* de *Ci*
    Mettre à jour les configurations avec *nd* ***venantDe*** *Ci*
    généraliser *nd* -> *fils(nd)*
    ajouter *fils(nd)* à *Ci*
  f**inFaire**
finTantQue
Calcul du résultat final

Sélectionner
Dépendant de la manière dont on désire parcourir l'espace de généralisation; dans notre application, le 1er trouvé.
Supprime le graphe de *Ci*.

Mettre à jour les configurations avec *nd* ***venantDe*** *Ci*
/ mise à jour de l'espace de généralisation/
Si $nd \notin G$,
alors ajouter *nd* -> {graphe d'origine de *nd*} dans *G*
sinon
  Si *nd* avait déjà été atteint par une autre branche pour le même graphe de départ
  alors
    Si le coût de *nd* est plus faible,
    alors remplacer l'ancien graphe par *nd* **finSi**
  sinon
/ mise à jour des configurations partielles /
  ajouter dans *G* le nouveau graphe d'origine de *nd* à la liste de ses graphes d'origine.
  supprimer toutes les configurations existantes contenant *nd*
  Pour tout *cp* = {a, b, c ...}->{x1, x2, x3, ...}$\in$ *Cp* **faire**
    si la liste des graphes d'origine de *cp* ne contient aucun des *x'i*, graphes origines de *nd*
    alors
    créer une copie de *cp*, ajouter *nd* à gauche et les *x'i* à droite :
    nouvelle configuration = {a, b, c,..., n}->{x1, x2, x3, ..., x'1,x'2, x'3}
    ajouter la nouvelle configuration à *Cp*
    finSi
  finFaire
  créer une nouvelle configuration avec *nd*
  finSi
finSi

Figure 5 : Algorithme de généralisation

Trouver les généralisations communes de deux graphes conceptuels, et qui plus est les généralisations

de plus de deux graphes, a fondamentalement une complexité exponentielle. L'utilisation des coûts pour arrêter la généralisation, plus que de maintenir la sémantique des graphes, réduit la taille de l'espace de recherche. Ce n'est toutefois pas suffisant pour rendre la méthode suffisamment efficace. C'est pourquoi nous utilisons quelques principes d'implémentation pour améliorer les performances de l'algorithme. Par exemple, si le coût pour trouver une généralisation commune entre le prédicat d'un graphe et le prédicat de chaque autre graphe est déjà plus élevé que le seuil autorisé, alors il est inutile d'inclure ce graphe dans le processus de généralisation. En utilisant ce principe, l'algorithme s'applique non pas sur les graphes à généraliser, mais sur les graphes pouvant être la première généralisation commune à deux graphes racines (cf. partie *Initialisation* de l'algorithme de la figure 5). Par ailleurs, les graphes sont indexés par leur prédicat et par leur nombre de relations et de concepts, de sorte que des comparaisons de graphes ne sont effectuées que si ces caractéristiques sont identiques.

Le but de l'algorithme est de produire toutes les descriptions possibles de l'ensemble de départ, l'ensemble des configurations. Une configuration est un ensemble de graphes conceptuels, généralisés ou non, qui partitionne les exemples initiaux, i.e. les graphes racines. Après application de l'algorithme, seuls les nœuds généralisant plusieurs graphes racines ainsi que les nœuds initiaux sont retenus, les autres nœuds, qui ne sont la généralisation que d'un seul graphe conceptuel de départ, n'ayant aucune utilité pour nous (dans le cas de l'exemple Figure 4, les nœuds retenus sont ceux figurant dans la première colonne du tableau 1). Le coût de généralisation de chaque nœud est le coût moyen des coûts de généralisation pour ce nœud, obtenus en fonction des différents chemins permettant d'atteindre ce nœud à partir de différents graphes racines. Trouver l'ensemble des configurations consiste à construire tous les sous-ensembles de nœuds tels que tout exemple initial est couvert une fois et une seule dans cette configuration. Chaque configuration est évaluée par la moyenne des coûts de ses éléments généralisés.

Nœuds	Coûts moyens	Origines	
c	(1+3)/2 = 2	x1	x2
b	(1+1)/2 = 1	x1	x2
d	(1+6)/2 = 3.5	x1	x3
e	(1+2)/2 = 1.5	x2	x3
x1	0	x1	
x2	0	x2	
x3	0	x3	

Ensemble des configurations = {[c, x3] (2), [b, x3] (1), [d, x2] (3.5), [e, x1] (1.5), [x1, x2, x3] (0)}

Tableau 1: Résultats du processus de généralisation

Le calcul des configurations est effectué au cours de la généralisation (cf. *mise à jour des configurations* dans l'algorithme). À chaque étape, les configurations partielles sont mises à jour dès qu'un graphe généralise au moins deux graphes racines.

S'il y a plus d'une configuration à la fin du processus, nous choisissons, en appliquant les critères suivants dans l'ordre, celle qui permet de généraliser le maximum de graphes racines, qui contient le minimum de graphes et enfin qui est de plus faible coût (autre que 0), considérant qu'elle est proche de la généralisation commune la plus spécifique. En effet, nous considérons que le plus important regroupement de graphes inférieur au seuil limite choisi constitue la meilleure description de l'ensemble d'événements initiaux. Dans l'exemple, ces critères amènent à choisir la configuration [b, x3] de coût moyen 1. Cette manière de choisir la meilleure configuration semble adaptée à notre application. Dans d'autres, peut-être faudrait-il réétudier l'ordre ou même chercher d'autres critères plus adaptés.

L'algorithme que nous avons présenté est un algorithme de généralisation de graphes conceptuels assez général pour pouvoir être utilisé dans d'autres applications. En toute généralité, la complexité du calcul est prohibitive si l'on cherche toutes les généralisations. Cependant, avec des heuristiques venant du domaine d'application et exprimées avec des valeurs numériques simples, il est possible d'élaguer la recherche d'une manière très importante pour ne trouver que les généralisations les plus informatives.

Il n'est pas aisé dans une telle application, fondée sur l'utilisation d'heuristiques qui réduisent l'espace de recherche de manière très dépendante de la base de connaissances, d'effectuer une démonstration formelle de sa complexité. En revanche, nous avons cherché à montrer l'utilisabilité et l'efficacité de la méthode par la réalisation de tests étendus.

5. Résultats

Le processus a été appliqué aux résultats de MLK, mais cela ne permettait pas une validation sur des données en quantité suffisante car une partie importante des traitements nécessaires à l'obtention des Unités Thématiques, unités contenant les ensembles de graphes à généraliser, reste encore manuelle. En effet, cela nécessite un niveau d'analyse syntaxique et sémantique des textes encore inaccessible à l'heure actuelle de manière automatique. C'est pourquoi, en vue de tester plus amplement notre algorithme de généralisation, nous avons été amenés à engendrer de façon semi-automatique un grand ensemble de jeux de tests. Nous avons fait en sorte que l'ensemble des données en entrée (treillis de types, graphes canoniques, graphes de tests) soient le plus proche possible des données réelles auxquelles nous sommes confrontés, aussi bien du point de vue de la forme que du fond et du volume. Bien évidemment, cette manière de générer les jeux d'essais est biaisée, dans le sens où nous générons des jeux qui ressemblent le plus possible aux données réelles auxquelles nous sommes confrontés. Mais notre algorithme de génération est suffisamment souple pour pouvoir être adapté à d'autres types de données représentables par des

graphes conceptuels sans contextes imbriqués. Nous allons présenter ci-dessous ces diverses données et analyserons par la suite les résultats que nous avons obtenus à partir de celles-ci.

5.1. Connaissances sur le domaine

Notre application utilise, comme nous l'avons explicité au paragraphe 2, des connaissances sémantiques sur le domaine : un treillis de types de concept, un treillis de types de relation et un ensemble de graphes canoniques.

Le treillis de types de concept, dont on peut voir un extrait Figure 7, a été conçu à partir du travail de Karim Chibout [Chibout 1998], lequel a créé une ontologie des verbes d'action et des concepts qui leur sont liés contenant environ 3000 concepts. Nous en avons extrait des parties que nous avons reliées entre elles de façon à obtenir un treillis assez régulier du point de vue de la profondeur et du facteur de branchement. Les 368 concepts extraits sont ainsi répartis dans un treillis d'une profondeur moyenne de 5,59 pour un facteur de branchement moyen de 2,75. Parmi les 128 verbes retenus, nous en avons sélectionné 8 pour les munir de graphes canoniques. Ces graphes canoniques sont formés en fonction des types retenus dans le treillis et définis de manière à produire des jeux d'essai homogènes, tels que les résultats globaux aient une signification. Il s'agit de : *FaireMourir, FaireEprouver, Diviser*, Casser, *Voler*, Absorber, *Manger* et *Boire*. Les sous-types héritent du graphe canonique de leur père. La figure 6 montre le graphe canonique de *Boire*. Le concept *Fonction* est le sur-type de 55 concepts parmi lesquels on trouve *Artisan*, *Tueur* ou *Député*. *Liquide* est le sur-type, entre autres, de *Eau, Alcool* et *Sirop*.

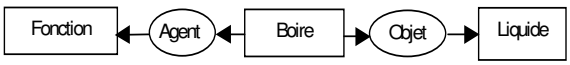

Figure 6 : Le graphe canonique de Boire

Le treillis de types de relation se limite dans notre application à un treillis plat : les relations ont toutes le même sur-type (T) et le même sous-type (⊥). Pour nos tests, nous avons utilisé quatre relations : *Relation, Agent, Objet* et *Lieu*.

5.2. Algorithme de génération des jeux d'essais

Disposant de l'ensemble des connaissances préalables nécessaires à nos tests, nous avons engendré des ensembles de graphes possédant des propriétés contrôlables pour la généralisation. Nous avons spécialisé des graphes canoniques dans des directions différentes tirées aléatoirement et sur des distances variées. Le but de notre application est alors de retrouver de façon non supervisée les graphes de départ ou, mieux, des spécialisations éventuelles de ceux-ci pouvant ne pas être apparus durant la phase de génération. Nous allons détailler dans les lignes qui suivent l'algorithme de génération des ensembles de tests.

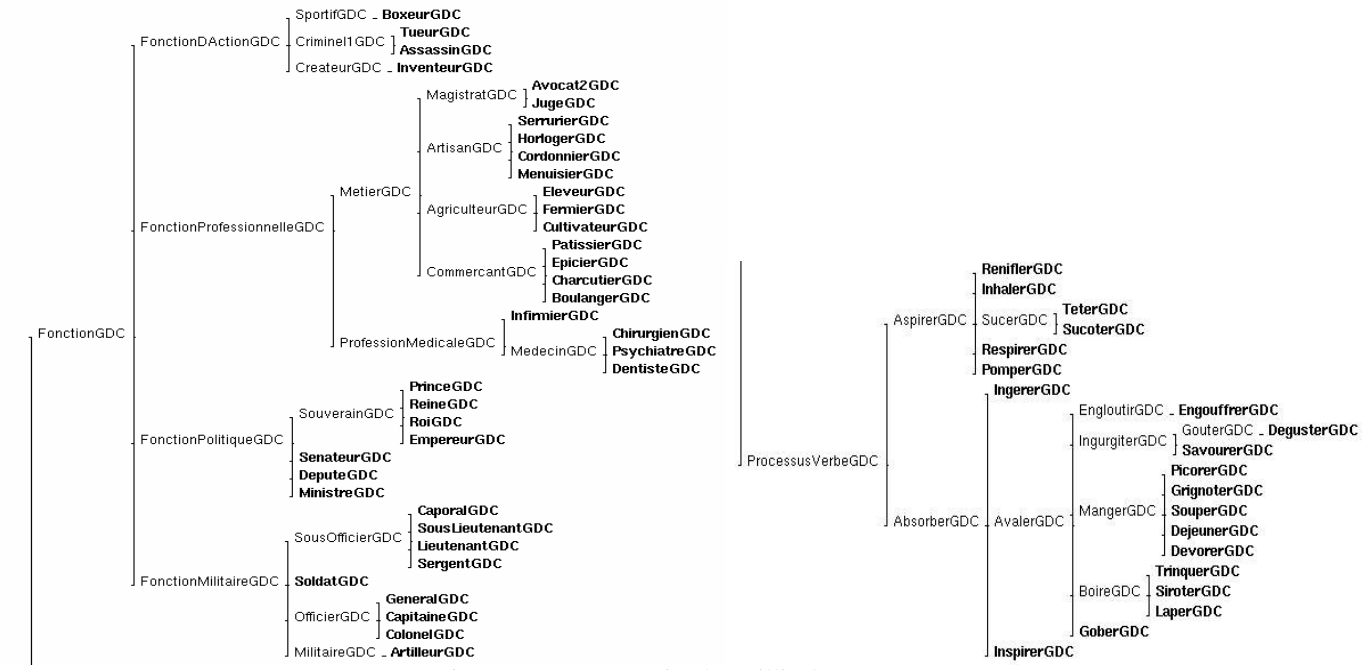

Figure 7 : Deux extraits du treillis de concepts

Nous voulons donc obtenir des ensembles de graphes tels que, pour chacun, il existe une partition des graphes où chaque partie (dénommée famille) se généralise en un graphe unique qui ne soit pas obligatoirement leur graphe canonique. Le moyen d'y parvenir consiste à tirer aléatoirement *n* prédicats, chacun à l'origine d'une famille. Le graphe canonique de chacun des *n* prédicats est ensuite spécialisé *s* fois pour produire *n* graphes, appelés graphes ancêtres. Chacun de ces *n* graphes est spécialisé *k* fois de plusieurs manières, créant *m* spécialisations différentes appelées graphes cousins. L'entrée du processus de généralisation est donc l'ensemble de tous les graphes cousins appartenant à diverses familles, le but étant de retrouver l'ensemble des graphes ancêtres.

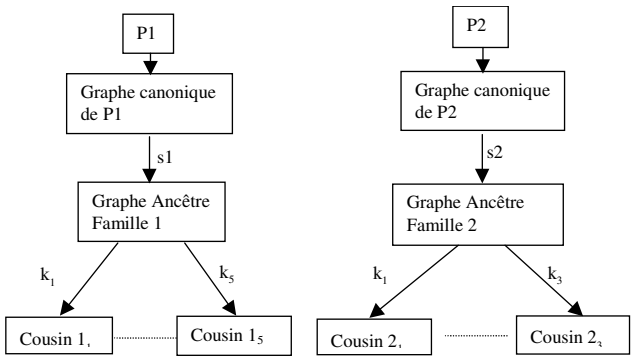

Figure 8 : Schéma du tirage d'un jeu d'essai

La figure 8 représente schématiquement un tel tirage. Deux prédicats, P1 et P2, sont à l'origine de deux familles. Leurs graphes canoniques sont spécialisés respectivement *s1* et *s2* fois pour obtenir les graphes ancêtres des deux familles. Le graphe ancêtre de la famille 1 est ensuite spécialisé $k_1, k_2, ...k_5$ fois, produisant 5 cousins dans la famille 1, tandis que le graphe ancêtre de la famille 2 est spécialisé $k_1, k,$ et k_3 fois pour obtenir une famille 2 contenant 3 graphes. L'entrée du processus de généralisation sera alors l'ensemble de tous les graphes $Cousin_{i,j}$. Le nombre d'opérations de spécialisation appliquées (restriction de type appartenant au graphe canonique ou non, ajout de relation) est précisé au moment de la constitution du jeu d'essai.

On peut voir ci-dessous un exemple de jeu d'essai contenant 4 graphes répartis en deux familles de deux graphes chacune. Les graphes ont été obtenus de la manière indiquée dans le tableau 2, ce qui produit les quatre graphes suivants :

Graphe 1:
```
[Grignoter]{
    – (Agent) -> [Infirmier],
    – (Objet) -> [Gâteau],
    – (Rel) -> [Pomme]}
```
Graphe 2:
```
[Dévorer]{
    – (Agent) -> [Infirmier],
    – (Objet) -> [Viande],
    – (Rel) -> [Riz]}
```
Graphe 3:
```
[Tuer]{
    – (Objet) -> [Reptile],
    – (Agent) -> [Assassin],
    – (Rel) -> [Voiture]}
```
Graphe 4:
```
[Noyer]{
    – (Agent) -> [Tueur],
    – (Objet) -> [Insecte],
    — (Rel) -> [Eau]
```

Prédicats	Graphes canoniques	Spécialisations pour ancêtres		Graphe numéro	Spécialisations pour cousins		
		du prédicat	*des concepts du graphe canonique*		*du prédicat*	*des concepts du graphe canonique*	*ajout de concepts*
Manger	**[Manger]** -> (Agent) ->[Fonction] (Objet) -> [Aliment]	0	3	1	1	1	1
				2	1	1	1
FaireMourir	**[FaireMourir]**-> (Agent) -> [Fonction] (Objet)-> [Vivant]	0	3	3	1	2	1
				4	1	2	1

Tableau 2: Détails de la génération d'un jeu d'essai

Cet exemple a provoqué la construction de 6 configurations dans un temps total de 115 s. On peut voir ci-dessous la solution optimale obtenue en 15 secondes suivie d'une autre non-optimale. Les coûts des opérations sont ceux présentés §5.4

Solution optimale :
```
[Manger]{
    – (Agent) -> [Infirmier],
    – (Objet) -> [Aliment]}
Coût : 8.0
[FaireMourir]{
    – (Agent) -> [Criminel],
    – (Objet) -> [Animal]}
Coût : 10.0
```
Solution non-optimale :
```
[Manger]{
    – (Agent) -> [ProfessionMédicale],
    – (Objet) -> [Aliment]}
Coût : 10.0
[FaireMourir]{
    – (Agent) -> [Criminel],
    – (Objet) -> [Animal]}
Coût : 10.0
```

Si on reprend la définition d'une bonne solution donnée §4.2, la solution optimale dans cet exemple respecte les critères établis : maximum de graphes racines généralisés, minimum de graphes dans la configuration et coût plus faible en cas d'égalité de configurations. Dans la solution non optimale, le concept *Infirmier* des deux graphes origine de la première famille a été sur-généralisé en *ProfessionMédicale*. On a donc bien obtenu la meilleure solution pour un coût inférieur à celui de l'autre solution. Les 4 autres configurations sont de coût moyen plus faible, mais ne généralisent pas l'ensemble des graphes de départ.
Nous allons dans la suite décrire deux séries d'essais. La première sert à tester un cas difficile, avec des graphes très spécialisés et des coûts de généralisation faibles, ce qui entraîne la génération d'un très grand nombre de configurations possibles et des temps de traitement élevés. La seconde montre l'efficacité du système avec des coûts plus élevés.

5.3. Tests dans le cas limite

Dans cette première phase, nous avons produit 33 ensembles de 4 graphes, constitués de 3 à 5 concepts et appartenant à 2 familles. Ces graphes ont été produits en appliquant entre 3 et 8 opérations de spécialisation sur des graphes canoniques. Le seuil de la limite de généralisation a été fixé à 10, et une limite de temps d'exécution a été fixée à 1 heure par jeu d'essai. Pour cet ensemble de données, nous avons attribué des valeurs faibles aux coûts des opérations afin de tester le cas limite : les coûts par rapport au seuil limite de généralisation permettent de calculer tous les graphes ancêtres. Ces coûts sont les suivants :

1. généralisation d'un concept quelconque : 1,
2. généralisation d'un concept du graphe canonique : 1,
3. généralisation du prédicat : 2,
4. suppression de relation : 1.

Le processus de généralisation a trouvé en moyenne 32,15 configurations par jeu d'essai dans un temps moyen de 28 mn, 30% des processus de généralisation ayant été arrêtés en raison de la limite de temps. Les 70% de cas résolus l'ont été dans un temps moyen de 15 mn.

5.4. Utilisation des coûts

Pour montrer l'efficacité du choix des coûts, nous avons fait le même type d'essais, mais avec des coûts multipliés par 2. Nous avons engendré 60 jeux d'essai de quatre graphes chacun. Avec une limite de temps fixée à 1h, nous obtenons 12,2 configurations en moyenne par jeu et en 20 mn, nous en obtenons 11,9. Ainsi 97,5% des configurations atteignables en une heure sont atteintes en moins de 20 minutes. Les résultats qui suivent ont été obtenus avec cette dernière limite de temps : 96,7% des jeux d'essai ont au moins une solution, et 73,3% des jeux d'essai n'ont pas atteint la limite de temps.

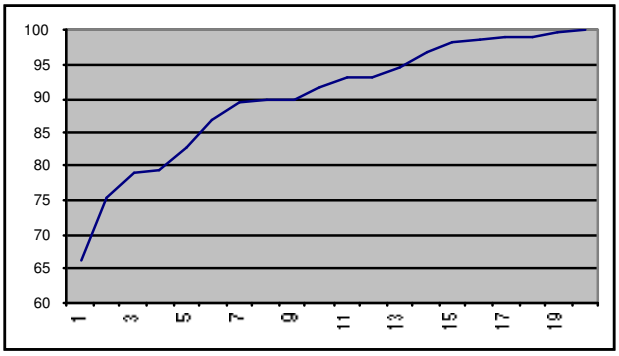

Figure 9 : Solutions obtenues en fonction du temps (en %)

La figure 9 montre le pourcentage de configurations obtenues, tous jeux d'essai confondus, en fonction du temps. Elle montre qu'en un temps raisonnable de 7 mn, 90% des solutions atteignables en 20 minutes, soit 92,3% de celles atteignables en une heure, sont déjà obtenues. Si l'on peut se permettre dans l'application de laisser de côté quelques solutions, alors on pourra utiliser la généralisation avec des temps d'exécution tout à fait raisonnables.
Pour ce type de test, la solution pertinente (i.e. informative) est donc celle qui comporte le ou les graphes généralisés égaux aux graphes ancêtres dont sont issus les graphes des jeux d'essai. Cette solution pertinente ne peut être trouvée que si, pour chaque graphe, la somme des coûts des opérations de spécialisation ayant servi à les obtenir à partir du graphe ancêtre est inférieure ou égale à la limite de généralisation, ici 10. Le nombre de jeux d'essais qui sont dans ce cas est de 22, soit 37%. Pour un temps d'exécution de une heure, la solution pertinente a été trouvée pour tous ces jeux d'essais, tandis qu'en 20 minutes on n'en trouve que 20, soit 90,9%. Parmi les 63% restant, pour lesquels le coût théorique de la solution optimale est supérieur à la limite, on trouve malgré tout une configuration possédant autant de graphes généralisés que de familles dans 15,8% des cas. Ceci s'explique par le fait que les graphes cousins sont engendré de façon aléatoire. Si par hasard, la spécialisation suit le même chemin pour plusieurs graphes, alors la solution pertinente effective se trouve être plus spécifique que le graphe ancêtre.

5.5. Analyse

Il faut noter que dans la première série de tests, les solutions pertinentes atteignables sont atteintes dans 70% des cas (23/33). Dans la deuxième série, 100% des solutions atteignables sont trouvées en une heure, mais elle contient un moins grand nombre de jeux d'essais. Ainsi, avec des coûts faibles, un grand nombre de solutions sont trouvées mais on en laisse aussi une partie de côté. En revanche, avec des coûts élevés, le nombre de solutions possible est plus faible mais le système les trouve toutes. Lors de l'utilisation du système sur des données réelles, il s'agira donc de moduler les coûts en fonction de ces deux extrêmes.
Ces essais montrent donc qu'un algorithme borné par des coûts permet de généraliser des graphes en un temps raisonnable, dans des applications pour lesquelles on peut définir ces coûts d'opérateurs de généralisation. Les

différences entre les deux séries de tests montrent l'intérêt de moduler le rapport entre les coûts et le seuil limite pour éviter d'engendrer la partie de l'espace de généralisation qui correspond à des sur-généralisations pour l'application, c'est-à-dire des graphes valides mais d'un niveau de description trop général, tout en produisant des généralisations intéressantes.
Démontrer la validité des solutions obtenues dans le cadre d'un apprentissage non supervisé n'est pas chose aisée. C'est ainsi qu'en définissant notre protocole de jeux de tests, nous avons fait en sorte de pouvoir caractériser la solution cherchée. Dans le cadre général de notre application, la validation pourrait être faite de manière externe par un utilisateur-expert où la question posée serait « la description produite vous semble-t-elle cohérente : événements distincts d'un niveau de description pertinent vis à vis de la situation décrite ? ». En cas de non-satisfaction, il est possible de relancer le processus en mettant en entrée les configurations partielles trouvées pour tenter d'obtenir de nouvelles généralisations satisfaisantes. Ou bien, la validation serait à effectuer via la validation d'un processus réutilisant ces connaissances.

6. État de l'art

Mineau [Mineau, 1992] et Bournaud [Bournaud, 1996] ont proposé un algorithme pour construire une classification à partir d'un ensemble initial de graphes conceptuels. Même si nous ne voulons pas classifier des données, trouver un niveau de généralisation peut être ramené à un problème équivalent puisque cela exige de calculer des généralisations possibles. La différence principale vient des opérateurs de généralisation utilisés. Ajouter la suppression de relation entraîne la nécessité d'une description complète de chaque graphe généralisé et non seulement d'une description partielle par les relations. Pour limiter la complexité effective, nous avons introduit des coûts associés aux opérateurs. Ces coûts doivent être fixés en fonction du type de généralisation que l'application exige. À l'instar de [Bournaud, 1996], nous tenons compte également de connaissances du domaine pour produire les graphes appropriés.
Notre travail se rapproche plus généralement des travaux effectués dans le domaine de la Programmation Logique Inductive [Muggleton et De Raedt, 1994] et plus particulièrement des travaux de Esra Erdem et Pierre Flener [Erdem et Flener, 1998], lesquels redéfinissent la généralisation minimale pour trouver un ensemble minimal de clauses généralisées en fonction d'un critère de surgénéralisation. Dans le cadre des graphes conceptuels, cette problématique se retrouve dans le travail de Marc Champesme sur la réduction de l'espace de recherche par l'utilisation de la notion de « subsomption empirique » [Champesme, 1995]. Dans ce dernier cas, les résultats obtenus ne s'appliquent qu'à des graphes non connexes alors que notre algorithme s'appuie sur des graphes simples connexes.
En ce qui concerne notre application, la formation de descriptions de situations prototypiques, notre travail diffère des travaux de [Lebowitz, 1983], [Mooney et DeJong, 1985], [Pazzani, 1988] et [Ram, 1993] essentiellement par le fait que nous ne voulons émettre aucune hypothèse sur l'existence de connaissances préalables sur les situations et mettons en œuvre un apprentissage totalement non supervisé. Le système MLK, qui concrétise cette application, permet de sélectionner les événements et les caractéristiques pertinents des situations rencontrées dans les textes, le système proposé dans cet article assurant la construction d'une description générale de ces événements et de ces caractéristiques.

7. Conclusion

L'apprentissage de structures afin de décrire des situations concrètes nous a amené construire le système MLK, capable d'identifier les événements relatifs à une même situation. À partir de ces résultats, dans le but de produire des descriptions de ces situations, nous avons étudié la généralisation d'événements, représentés sous forme de graphes conceptuels. Cependant, l'algorithme de généralisation que nous avons proposé est assez indépendant de notre contexte particulier pour pouvoir être utilisé dans d'autres applications.
Cet algorithme est entièrement programmé, en Smalltalk. Les essais réalisés montrent l'intérêt de mettre des coûts sur les opérateurs de généralisation dans des applications ne disposant que d'une théorie faible du domaine.
Le protocole de génération de jeux de tests que nous avons développé nous permet de contrôler divers paramètres tels que la définition d'une solution optimale, le nombre d'opérations de généralisation à appliquer pour trouver cette solution et l'homogénéité des jeux de test. Son intérêt est double : tester l'efficacité de notre approche et trouver la valeur à attribuer aux paramètres de l'algorithme par essais successifs.
Nous travaillons actuellement sur l'acquisition automatique d'ontologies à partir d'analyses syntaxiques partielles de phrases. Ces travaux vont nous permettre, à moyen terme, d'utiliser le processus de généralisation décrit sur de grosses quantités de données très similaires à des Unités Thématiques. À plus long terme, cela devrait permettre d'utiliser le système MLK complet sur des données extraites automatiquement de gros volumes de textes.

Références

[Bournaud, 1996] Isabelle Bournaud. Construction de hiérarchies conceptuelles pour l'organisation de connaissances. LMO'96, 1996.

[Champesme, 1995] Marc Champesme. Using Empirical Subsumption to Reduce Search Space in Learning. International Conference on Conceptual Structures, ICCS'95. Santa Cruz, California, USA, 1995.

[Chibout, 1998] Karim Chibout and Anne Vilnat, Computational Processing of Verbal Polysemy with Conceptual Structures, Actes Conceptual Structures : Theory, Tools and Applications (6th ICCS), Montpellier, LNCS volume 1453, p. 367-374, 1998.

[Ellis, 1992] Gerard Ellis. Compiled Hierarchical Retrieval. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.

[Erdem et Flener, 1998] Esra Erdem et Pierre Flener. A Re-Definition of Least Generalizations, and Construction Modes as a New Declarative Bias for ILP, Unpublished, 1998.

[Ferret and Grau, 1997] Olivier Ferret and Brigitte Grau. An Aggregation Procedure for Building Episodic Memory, 15th IJCAI, Nagoya, 1997.

[Ferret et Grau, 1998] Olivier Ferret et Brigitte Grau, Construire une mémoire épisodique à partir de textes : pourquoi et comment ?, RIA, vol.12, n°3, pp. 317-409, 1998.

[Lebowitz, 1983] Michael Lebowitz. Generalization From Natural Language Text. Cognitive Science, 7:1-40, 1983.

[Mineau, 1992] Guy W. Mineau. Induction on Conceptual Graphs: Finding Common Generalizations and Compatible Projections. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.

[Michalski and Stepp, 1983] K. B. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In Machine Learning, An Artificial Approach, Vol. 1, pp. 331-363, Morgan Kaufmann, 1983.

[Mooney and DeJong, 1985] Raymond Mooney and Gerald DeJong. Learning schemata for natural language processing. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985.

[Muggleton et De Raedt, 1994] Stephen Muggleton et Luc De Raedt, Inductive Logic Programming : Theory and Methods. Journal of Logic Programming 1994 :19,20 :629-679.

[Pazzani, 1988] Michael J. Pazzani. Integrating explanation-based and empirical learning methods in OCCAM. In Proceedings of the Third European Working Session on Learning, Glasgow, 1988.

[Ram, 1993] Ashwin Ram. Indexing, elaboration and refinement: incremental learning of explanatory cases. Machine Learning, 10(3):7-54, 1993.

[Sowa, 1984] John F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, 1984.