THÈSE D'UNIVERSITÉ

présentée à

L'UNIVERSITÉ de PARIS VI

pour obtenir le titre de

DOCTEUR D'UNIVERSITÉ ÈS INFORMATIQUE

par :

NATHALIE BEAUBOUCHER

Sujet de la thèse :

ANAÏS : RAISONNEMENT À PARTIR DE CAS EN RÉSOLUTION DE PROBLÈMES

Directeur de recherche François Rechenmann

Soutenue le 28 juin 1994 devant le jury composé de :

M. Jean Gabriel Ganascia, directeur de thèse
M. Joël Quinqueton, président et rapporteur
Mme Brigitte Trousse, rapporteur
M. Jean-François Perrot, membre du jury
M. Gilles Bisson, membre du jury

Remerciements

Je remercie Mr François Rechenmann de m'avoir si chaleureusement accueillie dans son équipe et de m'avoir fait confiance en acceptant d'être le responsable de ma recherche. Il a su me motiver et me soutenir dans mes moments très difficiles, je l'en remercie très vivement.

Je remercie Mr Jean-Gabriel Ganascia d'avoir accepté d'être mon directeur de thèse bien que ma recherche se soit déroulée à Grenoble. Un grand merci pour sa grande patience et pour sa confiance. Je remercie aussi Mr Jean-François Perrot, qui a accepté de faire partie du jury, et sans qui je n'aurais pu obtenir une bourse du ministère et donc n'aurais pu effectuer ce travail. Je remercie Joël Quinqueton et Brigitte Trousse de s'être intéressés à mon travail en acceptant d'être rapporteurs. Merci pour leurs remarques et leurs critiques pertinentes très instructives. Je remercie vivement Gilles Bisson, membre du jury, qui à peine installé dans nos locaux, a fait un travail de relecture et de critiques impressionnant. Merci à lui de ses conseils, de sa grande disponibilité et de sa grande ouverture d'esprit et de cœur.

Je remercie de tout mon cœur Hervé Raynaud qui, grâce aux stages de communication si agréables et si riches, a su déclencher en moi une restructuration personnelle importante et m'a permis ainsi, de remettre en cause mon système relationnel. Sa grande générosité, sa grande gaieté et son naturel m'ont beaucoup aidée à retrouver tous les éléments positifs présents en moi et à les faire partager.

Je remercie Patrice Uvietta dont les nombreuses relectures et critiques m'ont aussi beaucoup aidée dans la rédaction de ce mémoire ainsi que dans la rédaction d'articles. Je le remercie de sa grande disponibilité, j'apprécie beaucoup son esprit accueillant et ses qualités de communication. Je remercie Danièle Ziébelin qui m'a toujours soutenue dans mon travail, grâce à ses critiques très constructives, j'ai pu très souvent éclairer beaucoup de points brumeux et imprécis. Je remercie Pierre Fontanille pour ses nombreuses relectures et sa grande patience.

Je remercie toute l'équipe de François Rechenmann dans laquelle il a souvent régné une ambiance aussi sympathique que travailleuse : Bruno Orsier pour sa grande sympathie et son aide « en ligne » relative aux problèmes LaTeX et autres problèmes « durs », Jerôme Gensel pour son grand humour, Pierre Girard pour ses blagues qui réchauffent le cœur et pour son précieux soutien surtout dans la phase finale très stressante pour moi, merci à lui d'avoir de bons réflexes « anti-humides ». Je remercie Florence Lemaire et Cécile Capponi et François Bremond pour les sympathiques réunions d'équipe en parallèle, Nina Tayar pour sa grande disponibilité et son cœur toujours ouvert, Pierre Fontanille toujours très sympathique, Olivier Schmeltzer dont la grande amitié m'a toujours réchauffé le cœur et remonté le moral, Sylvain Mahé travailleur du soir très motivé et très motivant, Jean-Marc Gabriel pour sa joie de vivre très communicante, Elsa Collin d'un dynamique peu commun et très agréable, Sophie Rouzier, Jutta Willamowski, Christian Chevalier, Jérôme Euzenat, Isabelle Crampe, Michèle San-Martin, Danièle Herzog pour leur sympathie.

Je remercie l'équipe des médecins électromyographistes Annick Vila, Françoise Reymond et Linda Abaoub pour leur enthousiasme et leur grande patience.

Je remercie ma mère, ma sœur Marie-Paule pour leur soutien et leur aide le jour de la soutenance. Je remercie aussi tous les membres de ma famille, mes nombreuses sœurs, mon frère, ma tante et mon oncle « d'Amérique », qui ne sont pas déplacés mais qui ont néanmoins pensé à moi.

Je remercie enfin tous mes amis qui m'ont soutenue durant ma thèse : Bruno Mazier pour sa grande générosité et sa bonté d'âme sans limite et dont l'aide finale m'a été particulièrement utile, Bruno Courcelles, Denis Morand et Jean-Paul Chemla si formidables, généreux et toujours présents dans les moments difficiles. Un grand merci particulier à Jean-Paul pour s'être déplacé spécialement à Paris, pour avoir supporté la circulation, la chaleur et la pollution Parisiennes et enfin pour m'avoir aidée à tartiner très rapidement avant la soutenance. Je remercie Jerôme Thomas pour son accueil toujours très chaleureux à Paris. Merci aussi à Jean-Louis Mazier qui m'a hébergée à Paris avec ma mère et ma sœur, sa gentillesse m'a beaucoup touchée. Philippe et Catherine Augerat dont l'amitié est difficilement descriptible, Philippe Bertrand pour son amitié partagée riche, sincère et tellement agréable. Je remercie tout particulièrement Philippe Garat, dont les discussions et les sourires m'ont apporté beaucoup de richesses, de bonheurs et de réconforts. Je remercie tous mes autres amis même si leur nom n'est pas cité, ils garderont toujours dans mon cœur une grande reconnaissance.

Table des matières

Table des figures

Résumé

Dans le cadre de l'aide à la résolution de problèmes, deux hypothèses sous-tendent notre étude. La première est qu'il existe une base de connaissances du domaine. Cette base de connaissances n'est pas forcément complète mais contient toutes les étapes de résolution de problèmes du domaine. La seconde hypothèse est qu'en résolvant un nouveau problème, l'utilisateur exprime une partie de la résolution à l'aide de la base de connaissances incomplète. Notre système propose alors le problème le plus similaire au problème à résoudre relativement aux étapes de résolution.

Retrouver des problèmes similaires dans le domaine est la problématique centrale du « raisonnement à partir de cas » qui est une nouvelle approche en intelligence artificielle pour construire des systèmes à base de connaissances. Le principe de cette approche est basé sur une analogie avec l'élaboration de l'expertise humaine qui se fait par accumulation d'expériences successives et sur une remémoration des cas analogues pour résoudre un nouveau problème.

Cette remémoration est souvent basée sur une analogie dite de« surface », c'est-à-dire sur les caractéristiques des expériences et du problème à résoudre. Mais il n'est pas toujours possible de capter ces caractéristiques, une analogie « structurelle » est alors nécessaire. Nous avons réalisé ANAÏS (ANAlogical Intelligent System) dans lequel le mécanisme de mémoration des problèmes analogues est lié à la structure des cas, c'est-à-dire à leurs étapes de résolution. Notre système s'intègre dans un environnement de résolution de problèmes où un problème est modélisé sous forme de tâches et de sous-tâches.

La méthode utilisée dans ANAÏS s'articule en deux étapes. La première étape réduit le nombre de cas à examiner en sélectionnant les plus pertinents. La seconde choisit le cas le plus similaire. Cette dernière est complexe au niveau de l'élaboration de l'appariement et au niveau de la définition d'une mesure de similarité. Les algorithmes étudiés sont ceux des systèmes de reconnaissance de plans, dans les domaines de la conception et de la planification. Des mesures de similarité de domaines différents sont étudiées et une mesure de similarité est proposée. L'application abordée se situe dans le domaine du diagnostic médical.

Chapitre 1

Introduction

Reproduire le raisonnement humain est une préoccupation majeure en Intelligence Artificielle (IA). Dans les domaines d'expertise, où sont nécessaires un grand savoir et un grand savoir-faire, l'IA a proposé des solutions. La résolution de problèmes, notamment dans les domaines difficilement formalisables, est un point central de l'IA et est à l'origine de la plupart des recherches engagées. Des systèmes dits « experts » se sont fortement développés à la fin des années 70. Ces systèmes se sont principalement attachés à modéliser les connaissances du domaine et à reproduire les décisions prises par l'expert. Cependant, en pratique, la plupart d'entre eux ont utilisé d'autres modes de raisonnement que ceux utilisés par l'expert. De nombreux générateurs de systèmes à base de connaissances (terme plus général) se sont alors développés prenant en compte des représentations de connaissances différentes et des mécanismes d'inférence différents [Gan85, HK87, HRWL83].

Acquérir les connaissances est alors devenu un problème important mais ce processus s'est très vite avéré laborieux et la nécessité de l'automatiser a alors été primordiale. Deux courants ont pris naissance, l'Apprentissage Symbolique Automatique (ASA) qui est aujourd'hui un courant de l'IA fortement développé [Kod92], et l'acquisition des connaissances qui a donné naissance à la création de méthodologies d'acquisition des connaissances telles KADS [GW88] ou KOD[Vog88].

Après avoir décrit brièvement ces deux courants et avoir mis en valeur leurs points faibles et leurs points communs, nous exposerons divers rapprochements déjà existants et en cours d'étude. C'est dans ce cadre que l'approche « raisonnement à partir de cas » est abordée. En s'inspirant des méthodes du raisonnement à partir de cas et en ayant comme préoccupation la réutilisation de parties de résolution de problèmes, nous présentons ensuite notre application ANAÏS.

1.1 Réponse au problème de l'acquisition des connaissances : les méthodologies

La méthodologie KADS [HKL+89] a eu un grand succès en Europe. Dans KADS, différents types de tâches ont été inventoriées et, pour ces tâches, des modèles génériques reflétant leur structure inférentielle ont été construits et forment une bibliothèque. Ces modèles génériques sont constitués d'un ensemble de structures inférentielles élémentaires simples reliées les unes aux autres. Pour la construction du modèle il est possible de :

1. Sélectionner un modèle initial déjà existant dans la bibliothèque, qui sera alors nommé modèle d'interprétation, puis instancier ce modèle avec les connaissances du domaine.

2. Sélectionner plusieurs parties issues de différents modèles génériques, les combiner et ajouter le nouveau modèle obtenu dans la bibliothèque. Enfin, comme précédemment, on instancie ce modèle avec les connaissances du domaine.

3. Si aucun modèle générique ou aucune partie des différents modèles ne convient, construire un nouveau modèle, le rajouter à la bibliothèque, puis instancier ce modèle avec les connaissances du domaine.

Cette approche par les modèles génériques a inspiré la plupart des autres méthodologies développées [Aus89]. Cependant, des problèmes sont apparus au cours de l'utilisation de ces méthodologies comme celui de la sélection initiale du modèle ou celui de l'affinage du modèle. Aucune indication n'est en effet donnée sur la manière de choisir les modèles initiaux car cela dépend du type de problème traité mais aussi du domaine d'application lui-même. Répondant en partie à ce problème, il existe des travaux sur la caractérisation des problèmes [Cha83, Cla84, Cla85].

D'autre part, les modèles proposés sont beaucoup trop généraux pour s'appliquer directement à un domaine particulier et l'utilisateur doit affiner ces modèles pour qu'ils s'ajustent précisément à son type de problème. Cet affinage successif constitue un processus complexe et assez lourd. Sa mise en œuvre est un problème clef de la modélisation de la connaissance du domaine. Là encore, ce processus est à la charge de l'utilisateur et aucune aide n'est fournie.

Bien que la bibliothèque des modèles génériques de KADS s'enrichisse au fur et à mesure de son utilisation, l'automatisation de la construction de nouveaux modèles paraît indispensable. Cela d'autant plus que si ce processus est effectué en fonction de problèmes bien précis, le modèle construit sera forcément très détaillé, réduisant beaucoup la phase d'affinage.

C'est à partir de cette idée de génération automatique de modèles génériques, que nous avons pensé à mémoriser, pour un problème, toutes ses étapes de résolution. En effet, à partir d'un ensemble d'étapes de problèmes jugés similaires, une généralisation pourrait produire le modèle générique recherché.

Cette construction automatique répondrait donc aux deux problèmes importants de KADS : le choix du modèle et son affinage successif. Il reste cependant un intermédiaire indispensable : juger de la similarité des étapes de résolution de problèmes. Nous nous sommes donc tournés vers l'apprentissage symbolique automatique, qui utilise la plupart du temps cette notion de similarité.

Bien que l'ASA soit une approche tout à fait différente de l'acquisition des connaissances, il n'en reste pas moins que leur but initial est identique : élaborer des bases de connaissances. En ASA, bien que l'expert puisse ou doive intervenir à un moment donné du processus d'élaboration, cette intervention est minimisée. La non disponibilité des experts et le coût de leurs interventions sont un souci majeur, les techniques et les travaux en ASA ont donc pris une orientation différente.

1.2 Apprentissage symbolique automatique et acquisition des connaissances

Les travaux en ASA se sont attachés à produire des algorithmes soit pour transformer des connaissances déclaratives ou exemples, en connaissances procédurales, soit pour généraliser des exemples [Mic83] soit pour ranger des exemples dans des catégories [MS83, Mic93], soit enfin pour résoudre des problèmes par analogie [Kod92]. Bien que l'ASA soit une réponse à la difficulté de formulation des connaissances dans les systèmes à base de connaissances, il n'est pas toujours simple d'exprimer des connaissances complexes d'un domaine d'expertise dans une représentation formelle d'un système d'ASA. Des langages intermédiaires comme ceux utilisés dans les méthodologies pourraient s'avérer adéquats.

D'autres échanges sont aussi possibles entre les deux approches acquisition et apprentissage. Par exemple, des systèmes d'apprentissage récents comme APT [Ned92] prennent en compte de manière significative les compétences des experts pour contrôler l'apprentissage. Une interaction avec les experts existait déjà [SB93] pour valider les nouveaux concepts appris et pour corriger l'appartenance d'un exemple à un concept. Dans APT, les interactions avec l'utilisateur sont utilisées pour compléter le langage de description de concepts et pour réduire la sur-généralisation des exemples. D'autre part, des études sur la possibilité d'utiliser les méthodologies d'acquisition de connaissances pour formuler des exemples en ASA ont été menées [D'e91, HMH92].

L'intégration globale des deux approches est cependant difficile, citons pour exemple l'atelier de génie cognitif KEW [Cal90] où il existe un module d'apprentissage et un module basé sur une méthodologie. Mais la fusion des modules reste un problème car les représentations des connaissances dans les deux modules sont différentes. La construction de taxinomies est un objectif commun aux deux approches. En effet, plusieurs systèmes d'apprentissage symbolique construisent automatiquement des taxinomies [MS83], et certaines méthodologies ont un module d'acquisition des taxinomies comme, par exemple, KOD [Vog88] où la taxinomie est un élément majeur du modèle cognitif.

1.3 Vers une conciliation des deux approches

Comme nous l'avons vu précédemment, la représentation des connaissances diffère suivant l'approche utilisée. De plus, le choix d'une représentation de connaissances influence aussi le passage de la phase d'acquisition à la phase d'implémentation.

Les méthodologies d'acquisition des connaissances proposent des modèles conceptuels qui constituent un modèle intermédiaire entre la connaissance exprimée telle quelle par l'expert et les connaissances effectivement implémentées dans un langage ou système. Cette approche est justifiée par le fait que les langages ou systèmes informatiques sont de plus en plus complexes, il est donc nécessaire de créer des représentations de connaissances de plus en plus déclaratives, compréhensibles et maniables pour faciliter l'analyse du problème à résoudre.

Permettre l'expression de la résolution d'un problème à un niveau d'abstraction plus élevé est donc indispensable et les systèmes à base de connaissances de seconde génération répondent en partie à ce problème. D'autre travaux ont répondu à ce problème, notamment ceux sur les modèles de tâches [Cha87, SLSV84, HW87, Rou88, PGD91, UWZ93]. Ces modèles sont basés sur le concept de tâche, une structure de données qui permet de modéliser les opérations lors de la résolution d'un problème sous forme déclarative, encapsulant de cette manière les connaissances procédurales ou méthodes.

Quelque soit la représentation de haut niveau utilisée, dans certains domaines d'application, il n'est pas toujours possible de construire un modèle du domaine car les experts résolvent des problèmes uniquement grâce à leur expérience. C'est dans ce dernier contexte que nos travaux s'effectuent, à cette fin nous utiliserons et étendrons le raisonnement à partir de cas, qui est un type de raisonnement encore peu étudié en intelligence artificielle.

Outre cette différence de représentation en ASA et en acquisition des connaissances[1], examinons la modélisation du raisonnement dans chacune de ces approches.

[1] Pour plus de commodité, on notera AC par la suite

En acquisition des connaissances, différents types de raisonnement des experts ont été répertoriés [Vog89] :

- Le raisonnement déductif.

- Le raisonnement inductif par passage d'un cas particulier à un cas générique, appelée la synecdoque.

- Le raisonnement inductif par analogie où des analogies entre deux problèmes sont recherchées et où les solutions apportées au premier problème sont appliquées au second.

Les méthodes d'apprentissage symbolique automatique utilisent les mêmes types de raisonnement :

- Le raisonnement déductif dans la programmation automatique et dans certaines méthodes de généralisation [Nic87].

- Le raisonnement inductif dans beaucoup de méthodes de généralisation et dans les méthodes de regroupement conceptuel [Kod92].

- Enfin le raisonnement par analogie [CHL+91, Car83] dont est issu le raisonnement à partir de cas.

Parallèlement, en résolution de problèmes, il existe aussi trois modes de raisonnement [Car86] :

- La première approche consiste à sélectionner, affiner et instancier des plans généraux pour les appliquer au problème à résoudre, il s'agit donc d'une méthode déductive.

- La seconde approche consiste à rechercher des opérateurs à appliquer pour passer d'un état à un autre jusqu'à un état but désiré en explorant un arbre de recherche (méthode déductive).

- La troisième approche est la résolution d'un problème par analogie à un problème similaire. C'est une approche inductive puisque les connaissances obtenues par analogie ne sont pas des connaissances sûres.

C'est cette dernière approche qui a été la moins étudiée jusqu'à maintenant en IA. Notons que ce type de raisonnement est répertorié aussi bien en ASA qu'en AC et dans notre travail, nous allons nous y attacher.

La raison majeure de ce choix est que lorsque les experts ne possèdent pas de modèle du domaine, la résolution d'un problème ne peut pas se faire par choix et instanciation d'un modèle générique. Le raisonnement à partir de cas[2](raisonnement par analogie au sein d'un même domaine) propose de résoudre un nouveau problème en extrayant les solutions de problèmes similaires et en les appliquant au nouveau problème.

Nous nous proposons d'étudier une approche intermédiaire, dans laquelle une partie du domaine est modélisée mais est incomplète. La résolution d'un nouveau problème nécessite de retrouver des problèmes similaires, mais permet aussi d'enrichir la base de connaissances lorsque celle-ci ne couvre pas la résolution de ce nouveau problème.

Il s'agit donc en fait de retrouver les cas similaires au problème à résoudre, à prendre en compte le modèle partiel existant de ce problème à résoudre et de compléter ce modèle vis-à-vis des cas similaires proposés.

Afin de faciliter la résolution du nouveau problème, même lorsque celui-ci n'est pas couvert par la base de connaissances, nous proposons de réutiliser les étapes de résolution des problèmes mémorisés. En outre, cette mémorisation pourrait permettre, par la suite, la construction automatique de modèles génériques en utilisant les ensembles de résolution de problèmes similaires.

Notre étude implique la mémorisation complète de toutes les étapes de résolution d'un problème, ce qui constitue une différence essentielle par rapport aux systèmes de raisonnement à partir de cas et introduit donc des difficultés supplémentaires, comme nous le verrons par la suite.

---

[2]Pour plus de commodités on utilisera l'abréviation RBC. Et ceci, bien que RBC signifie ≪ raisonnement basé sur les cas ≫, terme incorrect se traduisant plutôt par ≪ raisonnement fondé sur les cas ≫. On peut aussi trouver dans certains articles [BS91], le terme ≪ raisonnement à partir de cas ≫ qui trouve son origine par analogie à l'apprentissage symbolique automatique où l'on parle d'apprentissage à partir d'exemples.

1.4 ANAÏS : présentation générale

Rappelons l'objectif principal : la réutilisation de parties de résolutions de problèmes pour d'une part, résoudre des problèmes non couverts par la base de connaissances et pour, d'autre part, compléter au fur et à mesure la base de connaissances. Pour atteindre cet objectif, le contexte de travail est présenté (chapitre 1). Nous travaillons dans l'environnement de résolution de problèmes SCAI (pour Scientific Computing in Artificial Intelligence) développé au sein de l'équipe SHERPA. Dans cet environnement, la résolution d'un problème se fait par décompositions successives du problème en sous-problèmes. Plus précisément, la démarche de travail est modélisée par des tâches et des sous-tâches.

Dans ce contexte, des objectifs plus précis permettant de retrouver des problèmes (ou plutôt des décompositions de problèmes) similaires sont exposés. La mémorisation complète des décompositions de problèmes consiste, dans cet environnement, à stocker l'ensemble de la décomposition tâches/sous-tâches de chaque problème.

L'aide à la résolution de problèmes, dans notre système, s'effectue en deux parties. La première partie recherche, à l'aide des caractéristiques des problèmes mémorisés et des caractéristiques du nouveau problème, un ensemble de problèmes similaires au nouveau problème. La seconde partie, plus complexe, consiste, à partir des décompositions des problèmes mémorisés extraits dans la première phase et d'une décomposition partielle supposée connue du nouveau problème, à apparier chacune de ces décompositions complètes avec la décomposition partielle du nouveau problème.

Afin de mettre en valeur les points communs et les différences entre le RBC et notre système ANAÏS (ANAlogical Intelligent System), le RBC est ensuite présenté (chapitre 2). L'accent est mis sur les processus d'extraction de cas, utiles à l'élaboration d'ANAÏS. Les approches du RBC utilisent souvent, dans l'extraction des problèmes similaires, une mesure de similarité de surface. Dans ANAÏS et plus spécialement dans la seconde phase, une mesure de similarité structurelle est nécessaire.

Une description générale du système est alors proposée (chapitre 3). Celle-ci comprend deux parties essentielles : l'extraction de cas similaires par rapport à un ensemble de caractéristiques et la sélection des meilleurs cas par appariement structurel des réseaux de tâches. Dans ce même chapitre, le détail de la première partie d'ANAÏS est exposé.

Ayant ainsi réduit le nombre de problèmes à examiner, nous étudions les difficultés inhérentes à l'appariement de schémas de résolutions de problèmes dans SCAI (chapitre 4). Il apparaît que des règles de simplification sont nécessaires. Puis un algorithme de recherche des tâches partagées par chacun des problèmes extraits et le nouveau problème est proposé (section 4.2.4).

À partir de cet ensemble de tâches partagées, il faut définir quel est le problème le plus similaire. Une mesure de similarité est alors nécessaire. Les mesures de similarité ont pris une place importante en IA, et pour notre système celle-ci est déterminante quant au choix du « meilleur problème ». Différentes mesures de similarité sont donc étudiées (chapitre 5) aussi bien dans le domaine du RBC, dans le domaine de l'apprentissage symbolique automatique que dans le domaine de la psychologie cognitive.

Après une comparaison des différentes mesures observées et une évaluation vis-à-vis d'ANAÏS, la mesure utilisée est décrite (chapitre 6). Cette mesure prend en compte la spécificité de l'environnement de résolution de problème utilisé en calculant l'importance d'une tâche vis-à-vis de sa place dans la base de connaissances.

Le domaine de l'électromyographie très étudié au sein de l'équipe est présenté ainsi que l'application d'ANAÏS dans ce domaine (chapitre 7). En conclusion (chapitre 8), les perspectives d'ANAÏS sont présentées.

Chapitre 2

Raisonnement à partir de cas et appariement de décompositions

Dans le RBC, on se remémore des expériences ayant des caractéristiques similaires. Se remémorer des décompositions complètes ou partielles de problèmes est une problématique différente. En examinant les définitions et les processus du RBC, nous nous positionnons par rapport à l'approche classique et nous introduisons un modèle du RBC adapté à nos objectifs.

2.1 Raisonnement à partir de cas : présentation

2.1.1 Introduction au RBC

La plupart des personnes améliorent la résolution de problèmes par leur expérience. Elles résolvent difficilement de nouveaux problèmes mais plus rapidement des problèmes déjà rencontrés ou similaires. De plus, elles se souviennent des erreurs commises et évitent de les reproduire. C'est à partir de ces différentes constatations que des recherches en IA ont été entreprises, donnant naissance au raisonnement basé sur les cas. Améliorer la performance d'un système et le compléter sont en effet des besoins majeurs et des recherches en apprentissage symbolique automatique se sont développées dans ce but [Kod92]. Par ailleurs, le RBC apporte des réponses aux problèmes d'acquisition dans les systèmes à base de connaissances. En effet dans les domaines spécifiques, il est souvent plus aisé d'exprimer un ensemble de problèmes résolus que la théorie complète du domaine.

Le RBC est né d'une étude psychologique de l'élaboration de l'expérience humaine. Une théorie de la mémoire et de son organisation a été étudiée [Sch82]. Schank propose une mémoire « conceptuelle » qui combine la mémoire sémantique, liée à la compréhension des phénomènes, et la mémoire épisodique qui stocke des épisodes et événements temporels [Sla91]. Schank considère ainsi que la mémoire est uniquement dérivée de l'expérience. Il propose une structure de la connaissance par les « MOPs » (paquets d'organisation de la mémoire), un MOP étant un méta-scénario (une scène générique) instanciable par différentes situations vécues. D'autre part, il fait l'hypothèse que l'expérience s'acquiert par indexation de nouveaux épisodes à l'aide des cas précédents dans des catégories thématiques. Le RBC repose essentiellement sur l'hypothèse que les nouvelles expériences nous rappellent des cas ou événements passés, ces connaissances associées peuvent guider notre comportement. L'organisation de la mémoire et la gestion de celle-ci représentent en effet le caractère principal de l'intelligence et les mécanismes sous-tendant le RBC sont fortement inspirés des processus de la mémoire.

Le succès du RBC, notamment aux États-Unis, est dû au fait qu'il facilite l'acquisition des connaissances. Exprimer directement des cas résolus est plus naturel que d'essayer de dégager des règles ou des modèles de raisonnement génériques qui sont généralement difficilement exprimables car élaborés au fur et à mesure de l'expérience.

Il répond donc en partie au problème de l'acquisition des connaissances où, face au goulet d'étranglement mis en évidence à la naissance des « systèmes experts », de nombreuses voies de recherche ont été entreprises. Des méthodologies sous-tendant le processus d'acquisition ont vu le jour. Donnons pour exemple la méthodologie KADS qui a été élaborée dans le cadre d'un projet Européen « ESPRIT » : elle se base sur la réutilisation et la combinaison de modèles génériques propres aux différents types de raisonnement dans des domaines différents [GW88].

De même KOD est une méthodologie d'acquisition des connaissances développée au sein d'un groupe industriel [Vog88]. Elle est destinée à être utilisée par des cogniticiens, ses bases conceptuelles étant des notions linguistiques et anthropologiques. Il apparaît cependant que les méthodologies et les systèmes d'acquisition des connaissances soient encore difficiles à utiliser et parfois lourds à mettre en œuvre, d'où un nouvel intérêt pour les techniques RBC.

Dans ANAÏS, la fonction principale recherchée est une aide à la résolution d'un problème. Cependant l'exploitation de la possibilité d'acquisition est envisagée pour faciliter la création de la base de tâches inhérente à un ensemble de problèmes, ainsi que pour la compléter ou éventuellement l'étendre à d'autres types de problèmes.

Un autre avantage principal du RBC est qu'il possède des facultés d'apprentissage de par son incrémentalité. En outre, les systèmes développés, qu'ils soient sous forme de prototype, d'outil opérationnel dans les laboratoires de recherche ou, récemment, sous forme d'outil logiciel commercialisé, semblent plus efficaces et plus facilement maintenables.

2.1.2 Définitions et description du processus RBC

Le RBC se définit par ses buts, mais aussi par les étapes qui le constituent. Le but principal est d'utiliser l'expérience acquise pour améliorer les performances du système. Mais il est possible aussi d'y voir un palliatif des mécanismes d'inférence existants qui peuvent s'avérer insuffisants. Christopher K.Riesbeck propose la définition suivante [RS89a] : *Retrouver un cas antérieur dans la mémoire, tenter de déterminer sa pertinence et décider de ce qu'il y a à faire en fonction du cas extrait.* Mais cette définition ne nous semble pas suffisamment détaillée. Carbonell propose de manière plus détaillée une définition de la résolution de problème analogique [Car86] :

La résolution de problème analogique consiste à transférer des connaissances à partir d'épisodes passés de résolution de problèmes aux nouveaux problèmes qui partagent des aspects significatifs avec l'expérience passée correspondante et à utiliser la connaissance transférée pour construire des solutions aux nouveaux problèmes.

Cette définition s'applique dans les travaux de recherche sur l'analogie, et ces derniers sont distincts des travaux RBC. En effet, l'analogie et le RBC sont souvent confondus, alors qu'en réalité, leur signification est bien distincte. La plupart des travaux en analogie se sont concentrés sur les analogies inter-domaines où la relation entre des cas répertoriés et le nouveau cas est abstraite. Dans le RBC, l'accent est mis sur des cas intra-domaine où une paire de cas se recouvrent sur beaucoup de caractéristiques dans un domaine bien particulier [Sei89]. De cette différence essentielle s'ensuivent des modifications tant au niveau des processus d'inférence liés à la mesure de similarité, qu'au niveau de la représentation des cas, qui peut être focalisée en RBC sur une modélisation particulière liée au domaine d'application.

D'autre part, la définition de Carbonell est liée à la résolution de problèmes, ce qui n'est pas forcément le cas en RBC. D'après Kolodner [KM92], il existe deux types différents de systèmes RBC : les résolveurs de problèmes qui adaptent des solutions passées pour résoudre un nouveau problème et les systèmes utilisant les cas pour évaluer ou justifier une nouvelle situation [KM92]. Ce dernier cas se retrouve par exemple dans le domaine de la justice où une sentence doit faire référence à d'autres cas analogues déjà rencontrés.

Par exemple, le programme JUDGE [Bai86] travaille dans le domaine des sentences criminelles. Il modélise un jugement qui détermine les sentences pour les personnes soupçonnées de crimes. L'entrée est une description du cas, incluant la charge, les événements qui ont eu lieu, et les statuts légaux vis-à-vis des crimes de cette nature, e.g. les différents niveaux d'emprisonnements permis et les conditions associées. Un nouveau crime est d'abord interprété en évaluant les événements importants, l'intentionnalité, la justification, etc... Ensuite le cas interprété est utilisé pour retrouver des cas similaires dans la bibliothèque. Puis, la sentence stockée dans le cas extrait est adaptée au crime courant en renforçant ou allégeant la sentence selon la comparaison effectuée avec l'ancien crime. Dans ce domaine où le bon et le mauvais sont difficiles à déterminer, il est important d'être cohérent avec les anciens cas, d'où le terme de justification utilisé par Kolodner [KM92].

La distinction résolution de problèmes/justification est cependant discutable, justifier une prise de décision ou une solution peut en effet être vu comme un problème en soi à résoudre, il s'agit donc aussi d'une sorte de résolution de problème. Pour proposer une définition adaptée au raisonnement par cas, nous garderons toutefois la différence entre résolution de problèmes et justification, qui sont distingués dans la plupart des écrits dans le domaine.

Nous proposons une modification de la définition de Carbonell, pour s'adapter au domaine du RBC qui devient :

Le processus de raisonnement par cas consiste à retrouver des connaissances à partir d'épisodes passés dans un domaine particulier, connaissances qui partagent des aspects significatifs avec l'expérience passée correspondante, et à utiliser les connaissances transférées pour construire des solutions aux nouveaux problèmes, ou pour justifier des solutions du domaine.

Dans le domaine du RBC, la modification d'une telle définition est dite modification « de surface », puisqu'elle se base sur les différences existantes entre l'analogie et le RBC. Le processus de modification, encore appelé adaptation, est, s'il existe dans le système RBC, souvent basé sur des différences de surface et des règles à appliquer pour chacune des différences observées.

Ce type d'adaptation se retrouve par exemple dans JUDGE [Bai86]. La sentence du cas extrait est adaptée au crime courant en renforçant ou allégeant la sentence selon la comparaison effectuée avec l'ancien crime. La phase d'adaptation, ou modification, correspond à une analyse différentielle comparant le cas courant avec les cas extraits de la mémoire. Les règles de modification de sentence sont cependant en nombre limité et sont assez approximatives. Par exemple, si les crimes sont différents et si les mobiles sont identiques, la peine d'emprisonnement sera diminuée ou augmentée de 25%. Il n'existe que six règles du même style où la sentence est modifiée de 25, 50 ou 75 %.

Dans le RBC, une des hypothèses de base est que l'intelligence n'est pas forcément liée à une grande compréhension des phénomènes, mais plutôt à une certaine habileté à indexer les informations dans la mémoire de manière à les retrouver rapidement lorsque cela est nécessaire. Cette hypothèse se retrouve dans les différentes étapes qui définissent le processus RBC. Citons pour anecdote, l'exemple de Riesbeck [RS89a], où un interlocuteur va chez la bibliothécaire et où il décrit précisément ce qu'il désire. La bibliothécaire a trois possibilités : indiquer où chercher le livre, se rappeler un livre qu'elle a lu et qui correspond tout à fait au besoin, ou encore indiquer un livre qu'elle n'a pas lu mais qui contient exactement ce qui est demandé. Si la bibliothécaire trouve exactement le bon livre, elle semble intelligente, mais en réalité, son intelligence consiste à traduire les données de l'interlocuteur dans le système d'étiquetage qu'elle s'est fabriquée, au fur et à mesure de son expérience, sur la connaissance de tous les livres.

La question se pose donc, la partie intelligente dans un système RBC n'est-elle liée qu'à la construction et la modification du système d'étiquetage ? Dans ce dernier cas, il est difficile d'affirmer qu'un système RBC est un système intelligent, car les systèmes d'indexation sont généralement introduits par le concepteur lui-même et les experts du domaine. Il existe cependant des systèmes RBC où le système d'indexation est introduit par le concepteur et se modifie automatiquement au fur et à mesure de l'expérience [BS91]. Ces systèmes sont en fait régis par un « méta-système » d'indexation qui agit sur le système d'indexation, mais le méta-système a été aussi introduit par le concepteur...

2.1.3 Étapes du RBC

Le RBC ne se propose pas de répondre à ces questions et se définit plus simplement par les étapes qui le constituent. Les soucis fondamentaux des systèmes RBC ne sont pas de se dire intelligents, mais de construire des raccourcis de raisonnement, d'éviter de reproduire les erreurs passées, de fournir parfois des explications ou des justifications et de faciliter l'acquisition des connaissances. Il s'agit donc d'un souci d'efficacité et d'amélioration progressive du système, tant au niveau de sa complétude que de son efficacité.

Le RBC met en jeu les opérations de base, décrites ci-dessous :

- Se rappeler les cas pertinents dans la mémoire en utilisant des caractéristiques du nouveau cas.

- À partir de la collection de cas retrouvés à l'étape 1, sélectionner le(s) cas le(s) plus prometteur(s) en utilisant des mesures de similarité.

- Construire une solution ou une interprétation pour le nouveau cas. Une solution est élaborée en adaptant les anciennes solutions.

- Tester et critiquer la sortie de l'étape précédente. Proposer des contre-exemples hypothétiques.

- Évaluer des résultats dans le monde réel et analyser ces résultats.

- Mettre à jour la mémoire en stockant le nouveau cas et les choix de critères permettant d'indexer le nouveau cas dans la mémoire. Cette étape est une étape d'acquisition de connaissances, il y a donc apprentissage et modification éventuelle du système d'indexage.

Stephen Slade a élaboré un diagramme représentant ces différentes étapes [Sla91]. Un cycle plus complet du raisonnement par cas a été élaboré par [Mig92] (figure 2.1).

De cette figure se dégage un certain nombre de problématiques centrales liées au RBC. Tout d'abord un point essentiel est de connaître l'organisation de la mémoire et la méthode d'indexation des cas. Repérer les cas dans la mémoire est en effet un processus essentiel à l'extraction de cas similaires.

Pour construire un système de raisonnement à partir de cas, deux autres problèmes restent à résoudre. L'un concerne le rapprochement de deux cas considérés comme similaires. Nous verrons que lorsqu'un cas est un schéma de raisonnement, les mesures de similarité semblent difficiles à définir. Le second problème est de modifier le cas similaire extrait, de manière à l'ajuster au nouveau problème.

Dans l'étape d'adaptation, la technique utilisée peut être structurelle où on modifie la solution. Elle peut être dérivationnelle (voir section 2.2), dans ce cas la solution mémorisée est adaptée, non pas en étant changée directement mais en ré-exécutant des parties du processus originel de la solution (cela suppose que l'on a mémorisé la séquence de planification qui a construit la solution). Dans l'étape de mémorisation, de la même manière, si ne sont mémorisées que les caractéristiques des cas, il s'agit d'une analogie transformationnelle. Si le raisonnement conduisant à la solution est mémorisé, on parle d'analogie par dérivation, car on recrée les lignes du raisonnement initial (voir section 2.2).

ANAÏS propose une partie de ce cycle complet, le mécanisme d'inférence étant lié au résolveur de problèmes SCAI et non pas au RBC

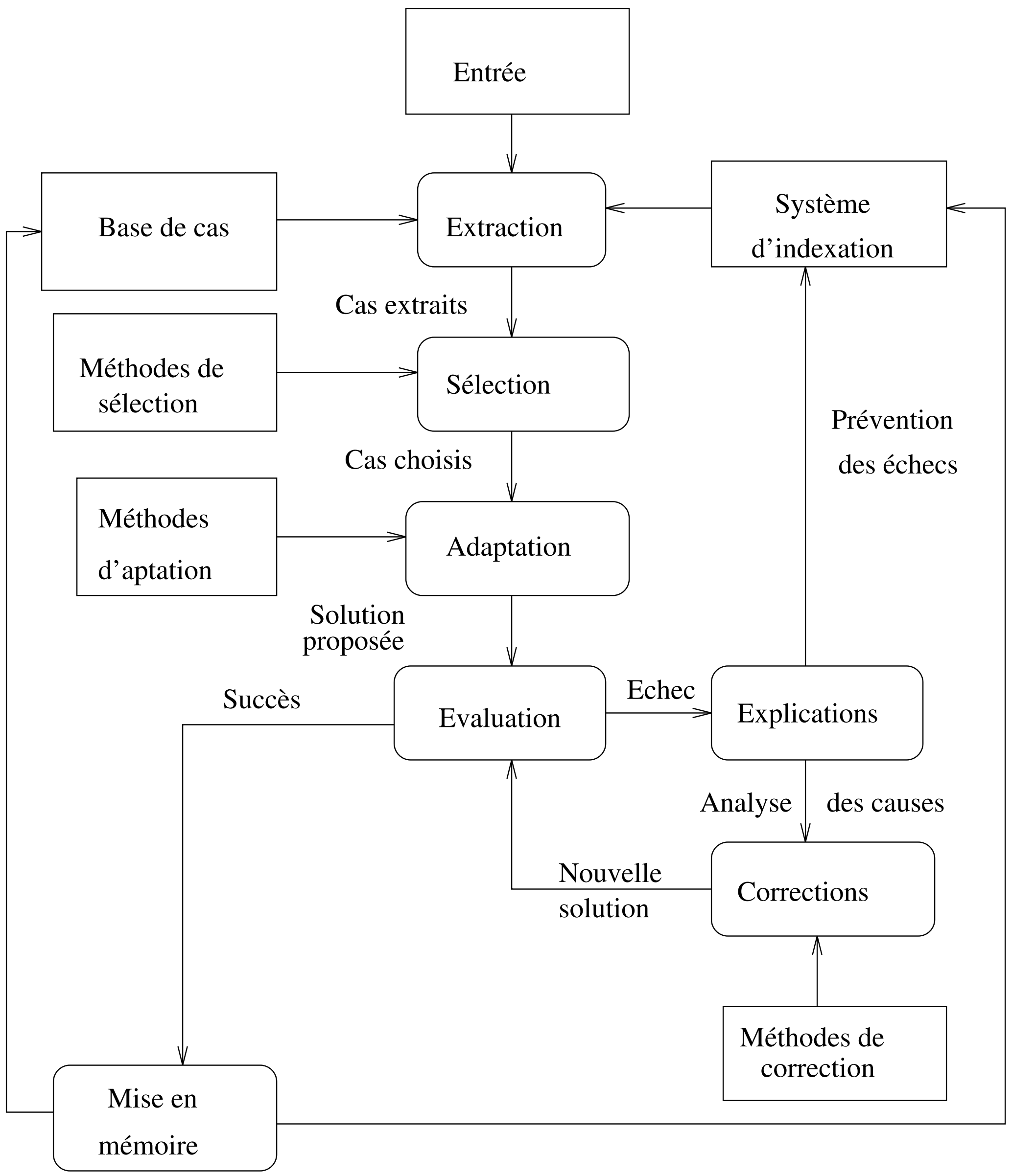


FIG. 2.1 - : Cycle général du RBC selon Hervé Mignot

lui-même, ce qui est généralement le cas des systèmes RBC purs (Figure 2.2).

Il existe cependant des systèmes où l'inférence analogique due au mécanisme RBC est couplée avec d'autres modes d'inférence. Le système DISCIPLE [TK93], par exemple, qui a pour but d'améliorer sa compétence et ses performances dans le contexte d'une théorie incomplète, combine différentes méthodes de résolution de problèmes : la décomposition d'un problème en sous-problèmes, la résolution de problèmes par contraintes et la résolution de problèmes par analogie. D'autres systèmes [OGA92] combinent des méthodes connexionnistes et des méthodes de résolution par analogie. La combinaison entre méthodes de résolution fondée sur des règles et celle sur le raisonnement par cas est aussi étudiée [RS89b]. Dans ANAÏS, il s'agit d'introduire la résolution de problèmes par analogie dans un environnement de résolution de problèmes fondé sur la décomposition d'un problème en sous-problèmes.

2.2 Approche de Jaime Carbonell

Dans [Car86] et plus récemment dans [VC91], Jaime Carbonell propose une méthode de résolution de problèmes appelée analogie dérivationnelle. La méthode de résolution de problèmes est une approche inductive par analogie. La résolution d'un nouveau problème se fait par recherche d'un problème similaire et par transformation de la solution de ce problème pour s'ajuster au nouveau problème. Cette approche semble être une réponse à notre problématique puisque d'une part, il est nécessaire de mémoriser toutes les étapes de raisonnement et que d'autre part, le processus de raisonnement d'un problème à résoudre est partiellement explicité et l'on cherche à retrouver des problèmes similaires. L'approche en question est celle du RBC en général, mais son originalité est de reconstruire une solution en recréant des schémas complets de raisonnement. Nous nous attacherons à décrire cette méthode de reconstruction et nous dégagerons ensuite les apports éventuels pour notre système.

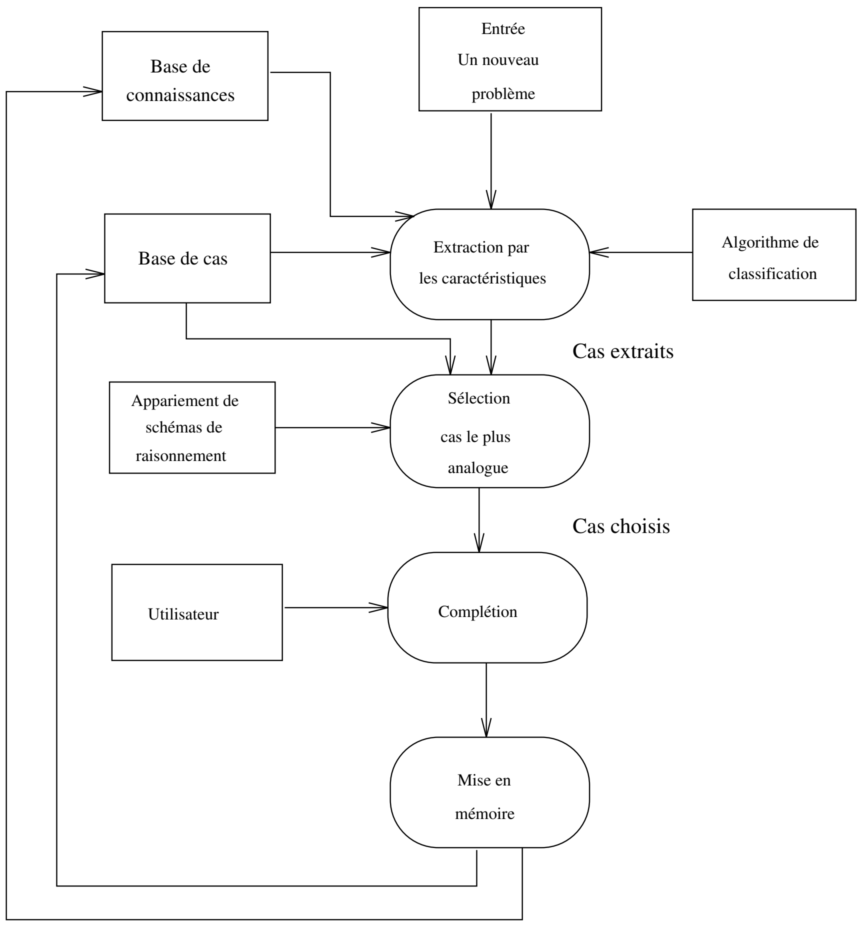


FIG. 2.2 - : Cycle du RBC dans ANAÏS

2.2.1 Introduction à l'approche de Carbonell

Pour clarifier son approche par analogie dérivationnelle et la distinguer des processus d'analogie transformationnelle, Carbonell distingue bien les deux types d'analogie :

- L'analogie transformationnelle où la sélection des expériences passées est effectuée en fonction d'une mesure de similarité. C'est le cas du système OGRE décrit dans les parties 3.2 et 5.4.2. Le processus de transformation est le suivant : deux problèmes partagent, selon la mesure de similarité, certains aspects en commun s'ils s'apparient selon un seuil prédéterminé. La connaissance transférée est la séquence d'actions de la solution retenue et le processus de transfert de la connaissance est obtenu en copiant l'ancienne solution et en la modifiant incrémentalement selon des pas de transformation primitifs jusqu'à résoudre le nouveau problème.

- L'analogie dérivationnelle où la connaissance extraite est l'ensemble du processus de raisonnement utilisé pour construire des solutions. Lors de la résolution du problème, chaque pas de raisonnement menant à la solution est mémorisé avec les informations suivantes : la structure des sous-buts du problème, chaque décision prise ainsi que des pointeurs sur la connaissance du domaine utilisée. Pour appliquer une solution extraite à la situation courante, il faut, pour chaque étape, commencer immédiatement après le début de raisonnement identique et vérifier si les justifications d'exécution de cette étape sont encore valides. Après qu'un schéma complet de raisonnement ait été trouvé pour s'appliquer au nouveau problème, il faut mémoriser ses différences par rapport à la solution initiale et modifier la solution en conséquence. L'idée de mémoriser des dépendances causales et de vérifier chaque pas de raisonnement passé est fortement inspirée de TMS [Doy79].

Carbonell utilise essentiellement l'analogie dérivationnelle et c'est cette approche que nous allons décrire par la suite. Mémoriser toutes les étapes du raisonnement ainsi que toutes les justifications de ces étapes peut paraître fastidieux et coûteux en place mémoire. Cependant, dans certains domaines, il n'est pas possible de construire des solutions à un nouveau problème sans toute cette connaissance. L'exemple donné par J. Carbonell est celui de la programmation automatique [VC91, Der86]. Dans l'élaboration d'un programme, la technique du raffinement progressif consistant à passer successivement de spécifications abstraites dans un langage de haut niveau à des descriptions de plus en plus opérationnelles jusqu'aux instructions dans le langage cible de l'ordinateur, est rarement utilisée. La pratique commune est en effet de se rappeler les programmes similaires et de reconstruire le nouveau problème en suivant ses traces. Les justifications de chaque étape de raisonnement doivent aussi être mémorisées. En effet, l'implémentation d'un même problème dans deux langages différents comme LISP et PASCAL implique la mémorisation du contexte car les deux solutions sont en effet totalement différentes. Les programmes produits résultent non seulement d'une philosophie différente, mais ne font pas référence aux mêmes structures de données et dépendent des outils disponibles dans chaque langage. Les avantages de l'analogie dérivationnelle sont évidents dans de tels domaines où la transformation directe d'une solution s'avère très souvent inadaptée.

2.2.2 Description détaillée du processus d'analogie dérivationnelle

L'idée principale est donc de mémoriser le processus complet du raisonnement pour construire (ou dériver) des solutions au problème à résoudre. La méthode est décrite de manière générale sans faire référence à un modèle d'implémentation particulier :

1. Lors de la résolution d'un problème, les connaissances ci-dessous sont mémorisées :

   - La structure des sous-buts du problème.
   - Des informations relatives à chaque décision prise (soit pour exécuter une action, pour explorer de nouvelles possibilités ou pour abandonner un chemin particulier). Ces informa-

tions pouvant être :

  - Les possibilités considérées et rejetées.
  - Les raisons des décisions.
  - Les débuts d'une mauvaise direction prise avec la raison pour laquelle cette voie paraissait prometteuse et la raison pour laquelle elle ne l'est pas en réalité.
  - Les liens entre la décision précédente et la décision en cours.

- Des pointeurs vers la connaissance utilisée lors de la prise de décision.
- La solution résultante. Si celle-ci n'existe pas, mémoriser les solutions proches ainsi que les sous-buts non réalisés et les raisons pour lesquelles il n'ont pu l'être. Si la solution dépend d'hypothèses de départ, sauvegarder aussi ces hypothèses.

2. Lorsqu'un problème n'est pas encore résolu et qu'il n'est pas possible de le résoudre par instanciation directe d'un plan générique, analyser le problème en utilisant des méthodes dites « faibles » (recherche d'heuristiques, analyse par les fins et moyens...).

3. Si après l'analyse du problème, les étapes initiales de raisonnement sont identiques à celles d'un problème passé, alors retrouver l'ensemble complet du processus de raisonnement de ce problème.

4. Appliquer une dérivation (l'ensemble des schémas de raisonnement menant à la solution) au problème à résoudre : pour chaque pas de raisonnement commençant après le premier pas retrouvé analogue, vérifier si les contraintes d'exécution sont toujours valides. Si elles le sont toujours alors appliquer le pas de raisonnement et continuer la vérification du prochain pas à effectuer. Si elles ne le sont plus, évaluer au point critique les possibilités non-choisies et sélectionner la plus appropriée ou abandonner l'analogie et en choisir une mieux adaptée. Ou, si plusieurs chemin d'échecs sont mémorisés pour l'étape en cours, essayer d'en trouver ne menant pas à l'échec dans le contexte actuel.

5. Lorsqu'un grand ensemble d'étapes de raisonnement sont similaires entre le problème à résoudre et le problème extrait, mémoriser les différences. Le nouveau problème peut alors servir de base pour une autre analogie et servir d'exemple pour la construction de plans génériques.

L'autre idée originale de cette approche est de considérer qu'à partir de cas concrets, une certaine connaissance est graduellement compilée donnant naissance à des processus de raisonnement plus généraux. Ceci revient à dire, qu'en reproduisant les problèmes, le processus d'analogie dérivationnelle peut produire des plans généraux qui peuvent être instanciés directement. En utilisant les solutions d'un grand nombre de problèmes analogues, des opérateurs généralisés de séquence de plans peuvent être abstraits. Ce processus demande que la solution dérivée analogiquement forme un ensemble d'exemples positifs, et que les solutions non valables forment un ensemble d'exemples négatifs. Ces ensembles sont donnés à un moteur inductif qui abstrait un plan général à partir des aspects communs de ces solutions. Ensuite le plan généralisé peut être instancié directement ou affiné.

2.2.3 Conclusion

Dans l'approche de Carbonell, la mémorisation du processus complet de raisonnement est nécessaire. Cependant beaucoup d'informations sont mémorisées, comme les contraintes de chaque étape de raisonnement ainsi que ses justifications.

Dans ANAÏS, seul l'ensemble des pas de raisonnement sera mémorisé. D'une part parce il ne s'agit pas de reconstruire une solution à partir du processus de raisonnement mais de proposer les plus similaires à celui en cours. La construction de la nouvelle solution est laissée pour l'instant à la charge de l'utilisateur.

La différence essentielle réside dans le fait que dans l'approche de Carbonell, le repérage des problèmes analogues est effectué par les étapes initiales des problèmes en mémoire et du problème à résoudre. Il n'y a pas de mesure de similarité à proprement parler, celle-ci est en effet implicite.

Dans ANAÏS, il s'agit d'une « ressemblance » globale entre toutes les étapes des problèmes en mémoire et du problème à résoudre. D'où la nécessité de définir une mesure de similarité.

Afin de mettre en œuvre un algorithme adéquat à l'extraction de plans similaires, nous avons examiné quelques exemples de réutilisation de plans et pas seulement dans le domaine du RBC.

2.3 Réutilisation de plans en raisonnement à partir de cas

La programmation automatique, comme nous l'avons vu dans 2.2, est un domaine où pour résoudre un problème par analogie, il est nécessaire de mémoriser toutes les étapes menant au but à atteindre. La solution du problème, énoncé par le but, est l'ensemble de toutes les étapes permettant d'atteindre le but. Il en est de même dans le domaine de la planification ainsi que dans celui de la conception. Dans la conception, cependant, le but à atteindre n'est pas forcément connu et fait partie aussi de la solution.

L'étude de l'analogie dans ces différents domaines peut donc apporter une réponse à celui d'ANAÏS où nous avons décidé de mémoriser toutes les étapes menant à la solution. La contrainte principale dans ANAÏS est que l'analogie ne doit pas être fondée sur un ensemble de caractéristiques mais sur tous les schémas de raisonnement mémorisés.

Plusieurs systèmes RBC ont proposé une résolution de problèmes par analogie, et ce, dans le domaine de la planification. La mémorisation de toutes les étapes de raisonnement amène des points communs avec le problème à résoudre dans ANAÏS, à savoir le calcul de la similarité par rapport à toutes ces étapes.

2.3.1 PLEXUS : un système de planification par adaptation

Dans PLEXUS [Alt88], la planification est élaborée à partir de plans déjà expérimentés et réutilisés après adaptation dans des situations jugées similaires. L'originalité du système est d'utiliser deux types de connaissances différentes : des plans spécifiques riches en détails et des plans généraux plus flexibles. La connaissance nécessaire dans le système est riche : des plans généraux, des connaissances taxinomiques et des connaissances causales.

Comme dans l'approche de Carbonell, il n'y a pas de similarité entre cas. L'algorithme recherche dans la base de cas une situation équivalente à la situation initiale du problème à résoudre. Il s'agit donc d'un appariement complet entre deux situations. Puis le système essaie d'appliquer étape par étape les séquences d'actions de l'ancien plan en vérifiant que les justifications associées soient encore valables (utilisation de la connaissance causale).

Un autre point original du système est que lors de l'adaptation, si une action n'est pas applicable, elle est remplacée par une autre action applicable et ce, grâce à des taxinomies d'actions (la connaissance taxinomique).

Étant donné que PLEXUS se situe essentiellement dans la phase d'adaptation, aucune information ne nous est nécessaire pour l'élaboration d'un algorithme d'appariement de séquence d'actions. Dans les systèmes de réutilisation de plans, la méthode de recherche d'un cas similaire au problème à résoudre n'est le plus souvent pas liée aux séquences d'actions. L'appariement de graphes est rarement utilisé.

2.3.2 ANAGRAM, un système de planification par analogie

ANAGRAM (ANAlogical GRAph Match) [Coo91] travaille dans le domaine de la programmation automatique. ANAGRAM est un algorithme rapide d'appariement de graphes. L'implémentation étant faite sur une machine parallèle, plusieurs centaines de cas sont traitées en même temps et le temps de calcul est donc réduit par rapport à un algorithme séquentiel. Il y a trois étapes essentielles dans ANAGRAM : la sélection de base, l'appariement et la génération de l'inférence. La sélection de base consiste à retrouver tous les cas de la base partageant des caractéristiques importantes avec le nouveau problème. Dans cette phase (appelée phase d'extraction dans les systèmes RBC), la similarité repose donc sur une similarité de surface.

Une fois ces cas extraits, le système utilise l'information décrivant le nouveau problème pour inférer le nouveau plan. L'information du problème à résoudre se décompose en deux sous-graphes : la description de l'état initial et la spécification de l'état final du problème à résoudre. ANAGRAM recherche ensuite les cas dont les graphes s'apparient le mieux avec les deux sous-graphes décrits. Si le plan d'un cas sélectionné s'applique au nouveau problème, il est proposé. Sinon, le système essaie de fusionner plusieurs cas similaires.

Une des limitations de ce système est que les états initiaux et finaux doivent s'exprimer sous forme de graphes, ce qui est le cas en effet dans la programmation automatique. L'exemple qui est donné est celui où le problème à résoudre est le calcul de la racine cubique d'un nombre avec une précision e. Le cas similaire extrait est le calcul de la division d'un nombre par un autre avec aussi une précision e donnée. L'état initial ne s'exprime pas par un sous-graphe car il contient deux nombres réels pour le problème à résoudre (le réel dont on veut calculer la racine cubique et la précision e) et trois nombres réels pour le cas de la base (les deux réels que l'on divise et la précision e). L'état final, correspondant au plan de réalisation du cas, s'exprime cependant sous forme de sous-graphes (voir figure 2.3).

La problématique est donc différente de celle étudiée dans notre système. En effet, nous cherchons des analogies entre des schémas de raisonnement et non pas des analogies entre l'état initial et l'état final d'un problème. D'autre part, supposer que ces deux états puissent s'exprimer sous forme de graphes limite les domaines d'application.

Examinons cependant la méthode de recherche de graphes en commun dans ANAGRAM. On recherche tout d'abord, un nœud équivalent au premier nœud de l'état final du problème à résoudre parmi le cas observé. Quand un nœud équivalent est trouvé, il est mémorisé dans une séquence et les nœuds suivants sont examinés. S'ils sont identiques, la séquence appariée est augmentée, sinon, le nœud suivant du problème est recherché à son tour dans le graphe du cas et une nouvelle séquence est générée. Si une seule séquence est générée et qu'elle est complète, c'est-à-dire qu'elle contient tous les nœuds de l'état but du problème, alors le cas de la base est sélectionné et l'ensemble des étapes du raisonnement de ce cas est adapté. Si aucun cas de la base ne permet un appariement complet, les cas possédant les plus grandes séquences en commun sont adaptés. L'étape d'adaptation n'est pas décrite, mais a

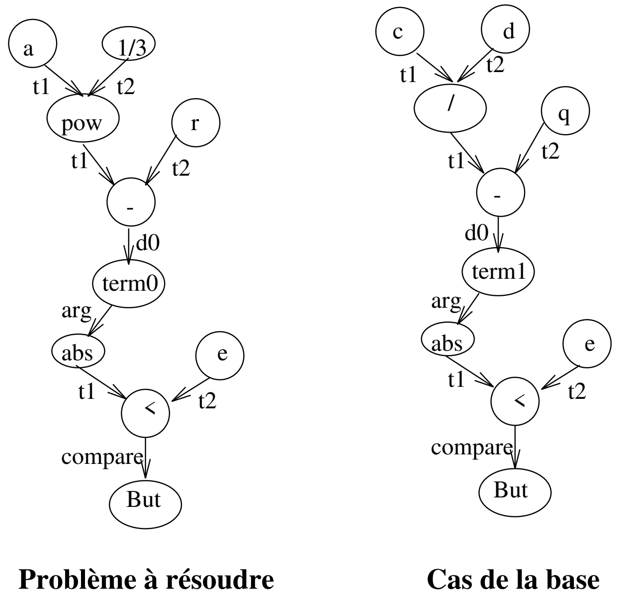


FIG. 2.3 - : Les sous-graphes de l'état but du problème à résoudre et de l'état but du cas extrait

pour originalité d'utiliser plusieurs cas si aucun appariement complet n'est trouvé.

La méthode utilisée est simple, aucun critère de valeur n'est lié aux nœuds, ce qui permet de sélectionner le cas possédant le plus de nœuds en commun avec le problème à résoudre. Par rapport à nos objectifs, nous retiendrons l'idée de conserver la liste des nœuds communs, ce qui nous permettra de sélectionner dans un premier temps les cas les plus « proches » au problème à résoudre.

2.4 Reconnaissance de sous-structures dans les scènes graphiques

La reconnaissance de sous-structures dans les scènes graphiques utilise aussi une représentation des connaissances basée sur les graphes et un appariement de graphes, c'est pourquoi nous tentons de déterminer son utilité par rapport aux objectifs d'ANAÏS.

Reconnaître des sous-structures identiques peut s'avérer intéressant pour notre étude dans la mesure où les images sont en général bruitées, ce qui laisse présumer un algorithme de mise en correspondance « flou ». Nous prendrons comme exemple le système SUBDUE [HJB92] où justement l'appariement entre deux scènes n'est pas un appariement strict mais tient compte des distorsions des images.

Dans l'algorithme de SUBDUE, les graphes sont utilisés pour représenter une situation géométrique. L'algorithme apparie la description d'une scène (image) avec des sous-structures existantes (carré, cercle, crayon, cube...). SUBDUE essaie de découvrir des nouvelles sous-structures en examinant différentes descriptions de scènes. L'originalité du système est de trouver des sous-structures qui ne sont pas forcément décrites de la même manière dans une scène ou une autre. Les sous-structures ont souvent des tailles et des orientations différentes. SUBDUE définit un ensemble de distorsions (transformations simples) possibles entre un sous-graphe et un autre sous-graphe. L'intérêt d'un tel système est de découvrir de nouveaux concepts ou de nouvelles sous-séquences et de les utiliser pour décrire de manière plus abstraite les autres exemples. Bien que les exemples soient tous issus de la géométrie descriptive, des applications possibles sont citées dans le domaine de la reconnaissance d'images, de la chimie et de la biologie moléculaire. Vis-à-vis d'ANAÏS, l'approche est intéressante car la recherche de sous-structures dans une structure englobante fait partie de notre problématique. Cependant, l'algorithme est loin de l'objectif d'ANAÏS qui est aussi de découvrir des sous-structures non pas identiques entre-elles mais « proches ». SUBDUE se concentre en effet sur une approche floue de l'analogie. De plus la sémantique est différente : il s'agit de scènes observées dans SUBDUE et de schémas de raisonnement identiques à découvrir dans ANAÏS.

D'autre part, étant donné que dans ANAÏS, il s'agit de rechercher une analogie parmi des étapes de résolutions, nous ne pouvons définir un ensemble de distorsions, les différentes étapes possibles de chaque problème mémorisé étant beaucoup trop nombreuses. Nous préférons donc nous baser sur un ensemble d'étapes communes, ce qui bien sûr repose sur un calcul de similarité tout à fait différent de celui utilisé dans SUBDUE.

2.5 Appariement de sous-buts dans CADET

CADET est un système proche du raisonnement à partir de cas puisqu'il se situe dans le domaine de l'analogie. Dans ce système aussi, on retrouve des points communs avec notre problématique : la mémorisation de séquences d'action et un algorithme d'appariement de graphes. Dans CADET cependant, nous verrons que l'appariement se fait par rapport à des sous-buts et cet appariement doit être complet.

Dans le but d'augmenter la créativité, Katia Sycara dans son système CADET [SN91a], [SN91b] utilise plusieurs cas dans des domaines différents pour concevoir de nouveaux objets. La constatation de base est que, lors de la création de nouveaux objets, il est possible de retrouver des sous-buts identiques au but du problème à résoudre ou à l'un de ses sous-buts. Une recherche de sous-buts communs entre les cas de la mémoire et le cas en cours de résolution est effectuée. Après avoir retrouvé les séquences d'actions répondant aux sous-buts du cas à résoudre, il reste à les combiner pour construire la solution.

L'exemple qui est donné se situe dans le domaine de la conception mécanique. Chaque système mécanique est décrit avec un ensemble de variables et un graphe d'influence entre les variables. Pour le problème à résoudre, sont connus : les paramètres du système et des graphes d'influence entre les variables. CADET procède alors en trois temps : tous les cas possédant les mêmes paramètres sont extraits, tous les cas possédant une des influences décrites dans le problème à résoudre sont extraits, puis, l'intersection entre les deux ensembles de cas précédents est faite. Notons que dans le second ensemble, les influences ne doivent pas forcément comporter les mêmes paramètres, une hiérarchie des influences existe par rapport au niveau de généralité des paramètres. À partir de ces cas extraits, le système vérifie chaque cas extrait vis-à-vis des graphes d'influence du problème à résoudre. Si le graphe (ou le sous-graphe) s'apparie complètement, le cas (ou le sous-cas) est sélectionné.

L'intérêt de la démarche de CADET ne réside pas dans l'algorithme d'appariement de graphes (qui par ailleurs n'est pas décrit et dont la complexité n'est pas réduite par rapport aux algorithmes classiques d'appariement de graphes qui sont NP-complets), mais dans la hiérarchie des graphes d'influence qui intègre des cas provenant de domaines différents. Les cas répertoriés se trouvent aussi bien dans le domaine de la mécanique des fluides (robinets, mélangeurs...) que dans le domaine de la mécanique des matériaux (balance...) ou que dans le domaine de signaux électriques.

Dans ANAÏS, la démarche est analogue à celle de CADET, dans le sens où plusieurs bases de tâches provenant de domaines différents peuvent être utilisées. L'utilisateur peut donc concevoir de nouveaux schémas de raisonnement qui réutilisent les schémas de raisonnement des cas de domaines différents. La différence importante réside dans le fait que l'appariement se fait sur des sous-buts et non pas sur la résolution complète d'un problème. D'autre part, l'appariement (via des passages vers la hiérarchie des paramètres) doit être complet et aucune mesure de similarité n'est calculée par rapport à cet appariement.

Chapitre 3

ANAÏS : description fonctionnelle et utilisation de caractéristiques

Après un rappel de nos objectifs, nous présenterons les principales différences avec le RBC. Une description globale du système est proposée et la première étape de sélection des problèmes par les caractéristiques est décrite en détail. L'étape originale dans ANAÏS d'extraction de cas vis-à-vis des étapes de résolution des cas est détaillée dans tout le chapitre suivant.

3.1 Objectifs et problématique

L'hypothèse de départ est qu'il existe déjà une base de connaissances du domaine, même si celle-ci est incomplète et l'utilisateur peut alors utiliser cette base pour exprimer partiellement la résolution de son nouveau problème.

Dans ANAÏS, nous travaillons dans un environnement de résolution de problèmes où un problème est décomposé en sous-problèmes par l'intermédiaire de tâches et de sous-tâches. Notre hypothèse revient donc à dire qu'il existe une base de tâches dans laquelle l'utilisateur peut choisir une tâche et la décomposer lui-même en sous-tâches. Cette base permet de résoudre un certain nombre de problèmes. Supposons que l'utilisateur essaie de résoudre un problème et s'aperçoive les tâches qu'il lancé ne lui permettent pas de le faire complètement avec la base actuelle.

Supposons qu'il connaisse quelques tâches utiles à la résolution de son problème, ANAÏS va grâce à ces quelques tâches, proposer des cas voisins, c'est-à-dire ayant aussi ces tâches dans leur résolution. L'utilisateur pourra étendre, en fonction de ces cas voisins, la base de tâches de manière à pouvoir résoudre le nouveau cas. Ainsi on améliore la base de connaissances en couvrant de plus en plus de problèmes, ce qui permet de compléter incrémentalement le système. L'utilisateur peut aussi ne plus savoir quelle tâche lancer pour résoudre son problème et un cas analogue peut lui rappeler comment faire. Il peut aussi se souvenir comment on fait et avoir trouvé une autre méthode pour le faire, il s'agit alors d'un enrichissement de la base de cas.

Cette problématique nécessite la mémoration[1]de problèmes analogues déjà résolus, elle concerne donc les travaux sur le RBC.

Dans l'environnement de résolution de problèmes utilisé (SCAI), des bases de connaissances décrivant des tâches à réaliser et l'enchaînement liant ces tâches entre elles sont actuellement opérationnelles. Citons en exemple la base IFREMER [PR89] qui permet de diagnostiquer des dysfonctionnements sur une plate-forme pétrolière en mer. Deux modes de fonctionnement sont possibles dans SCAI : le mode guidé où les tâches sont décomposées automatiquement par le moteur de SCAI et le mode libre où l'utilisateur peut lancer des tâches dans un ordre quelconque.

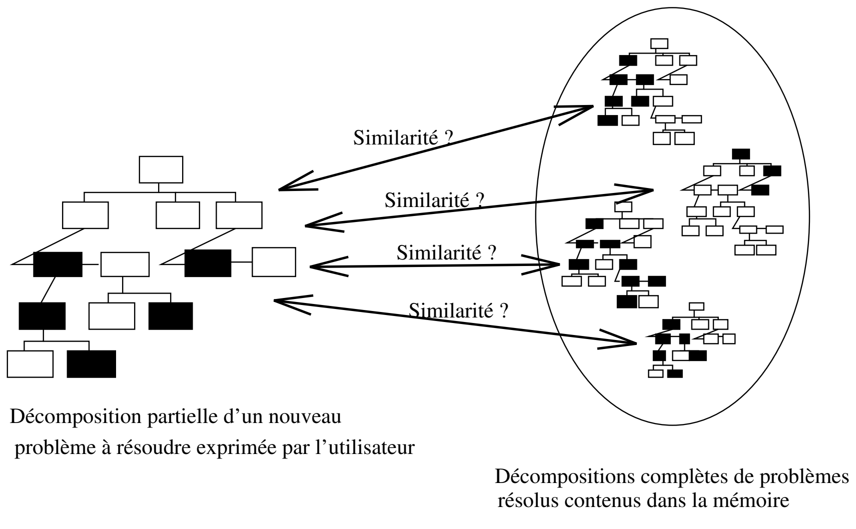
C'est grâce à ce mode que nous allons pouvoir aider l'utilisateur, soit pour résoudre totalement son problème, soit pour étendre la base de tâches de manière à résoudre un nouveau problème. Lorsque le problème peut être résolu par la base de tâches mais que l'utilisateur est perdu, un algorithme de reconnaissance de plans indique les tâches manquantes pour terminer le problème. Le cas qui nous préoccupe est celui où la

[1]Le terme de mémoration se distingue de la mémorisation. La mémorisation consiste à mettre en mémoire un événement ou un problème, tandis que la mémoration est le mécanisme permettant de retrouver un événement, une situation, un raisonnement qui est déjà contenu dans la mémoire. On trouve parfois le terme de remémoration, qui en effet plus évocateur et qui est en réalité le terme français, mais on utilisera cependant mémoration plus proche du vocabulaire anglais utilisé dans ≪ le Case-Based Reasoning ≫

base de connaissances est incomplète. La base de tâches ne peut décrire tous les problèmes du domaine, il est donc nécessaire de la compléter, voire de la modifier. Nous supposons cependant que certaines tâches seront utilisées, l'utilisateur connaît donc partiellement la résolution de son problème.

Le but d'ANAÏS est de proposer les cas analogues répertoriés dans la mémoire vis-à-vis du problème partiellement résolu décrit par l'utilisateur. Après complétion du problème à résoudre par l'utilisateur vis-à-vis des problèmes similaires proposés, le nouveau problème résolu est ajouté à la base de cas. Pour la complétion du nouveau problème, l'utilisateur pourra être amené à modifier la base de tâches en l'enrichissant de nouvelles tâches. L'affinage et l'enrichissement progressif de la base de tâches facilite l'acquisition des connaissances car celle-ci devient incrémentale.

En résumé, l'objectif principal est l'aide à la résolution de problèmes, dans le cadre de résolutions par décompositions successives (voir figure 3.1).



3.2 ANAÏS et le RBC : différences et originalités

Dans le RBC, le mécanisme d'inférence consiste à extraire les solutions des cas analogues et à modifier ces solutions pour les ajuster à la nouvelle situation (section 2.1.2). Dans ANAÏS, le mécanisme d'inférence doit prendre en compte celui de l'environnement de résolution de problèmes, c'est-à-dire le moteur de SCAI. Dans ANAÏS, il ne s'agit pas de donner la solution à un nouveau problème en fonction des différences observées (par adaptation), mais uniquement de retrouver des problèmes analogues. *A priori*, seules les phases d'extraction et de sélection seront utilisées dans ANAÏS. L'utilisation des cas analogues extraits est donc différente de celle utilisée couramment en RBC puisque nous ne cherchons pas à les adapter[2].

D'autre part, les analogies utilisées en RBC sont généralement des analogies dites « de surface ». Ainsi par exemple dans OGRE [Don89], les analogies se font en fonction de « facteurs-descripteurs» qui correspondent à des attributs et aux informations relatives à ces attributs. Les structures de données principales sont les cas exprimés à l'aide de facteurs. Les cas contiennent la description des expériences passées et sont stockés dans la mémoire d'expériences. Chaque cas est caractérisé par des facteurs qui expriment la description des caractéristiques du domaine. Un domaine est caractérisé à l'aide d'une collection de facteurs-descripteurs qui contiennent les informations suivantes :

- Nom du facteur, ex : température.
- Type pour le facteur, ex : nombre réel.
- Domaine de valeurs du facteur, ex : entre -20 et 300.
- La valeur par défaut.
- La méthode avec laquelle deux valeurs pour ce facteur sont comparées, ex : si T1 est $\pm$ 10 degrés de T2, considérer que c'est un bon appariement.

[2]Notons que les méthodes d'adaptation sont peu développées dans les systèmes RBC existants du fait de leur complexité.

- L'importance du facteur par rapport aux autres, ex : la température est aussi importante que l'humidité.

- Si nécessaire, la méthode pour utiliser les autres facteurs pour déterminer la valeur de ce facteur.

Pour déterminer la similarité entre deux expériences, le système OGRE examine chaque facteur commun aux expériences et les compare selon la méthode spécifiée dans le facteur-descripteur. Dans OGRE les fonctions de comparaison génériques sont connues, mais l'expert peut écrire sa propre fonction de comparaison et l'insérer dans le facteur-descripteur. Après avoir déterminé une valeur de comparaison pour deux valeurs d'un facteur, cette valeur est ajustée en appliquant le poids de ce facteur.

La plupart des analogies fonctionnent donc sur la base de « caractéristiques » [Tve77] des problèmes, mais l'expression de ces caractéristiques n'est pas toujours faisable dans tous les domaines, et est inadaptée dans le cas de problèmes complexes comme la planification ou la conception. L'originalité d'ANAÏS est donc de proposer des cas analogues, non pas en fonction des caractéristiques du problème mais en fonction de la structure du problème soit, en l'occurrence, toutes les étapes de la résolution.

L'approche de Carbonell prend elle aussi en compte la structure des problèmes mais l'analogie est basée sur des débuts de raisonnements identiques. Dans ANAÏS, nous proposons que l'analogie soit effectuée sur l'ensemble du processus de résolution et donc que les cas similaires soient des cas où plusieurs étapes de résolution sont identiques aux étapes de résolution du problème courant.

3.3 ANAÏS : description fonctionnelle

Le fonctionnement du système est organisé en trois phases indépendantes : une première phase chargée d'initialiser la base de cas, une seconde où sont décrites les connaissances du nouveau problème à résoudre et la troisième, effectuée par ANAÏS qui se charge de déterminer les cas analogues au nouveau problème par rapport à la base de cas actuelle. Dans ANAÏS, un cas comporte deux informations bien distinctes : un ensemble de caractéristiques et le schéma complet du raisonnement décrit sous forme de tâches et de sous-tâches. Le schéma global d'ANAÏS est décrit dans la figure 3.1.

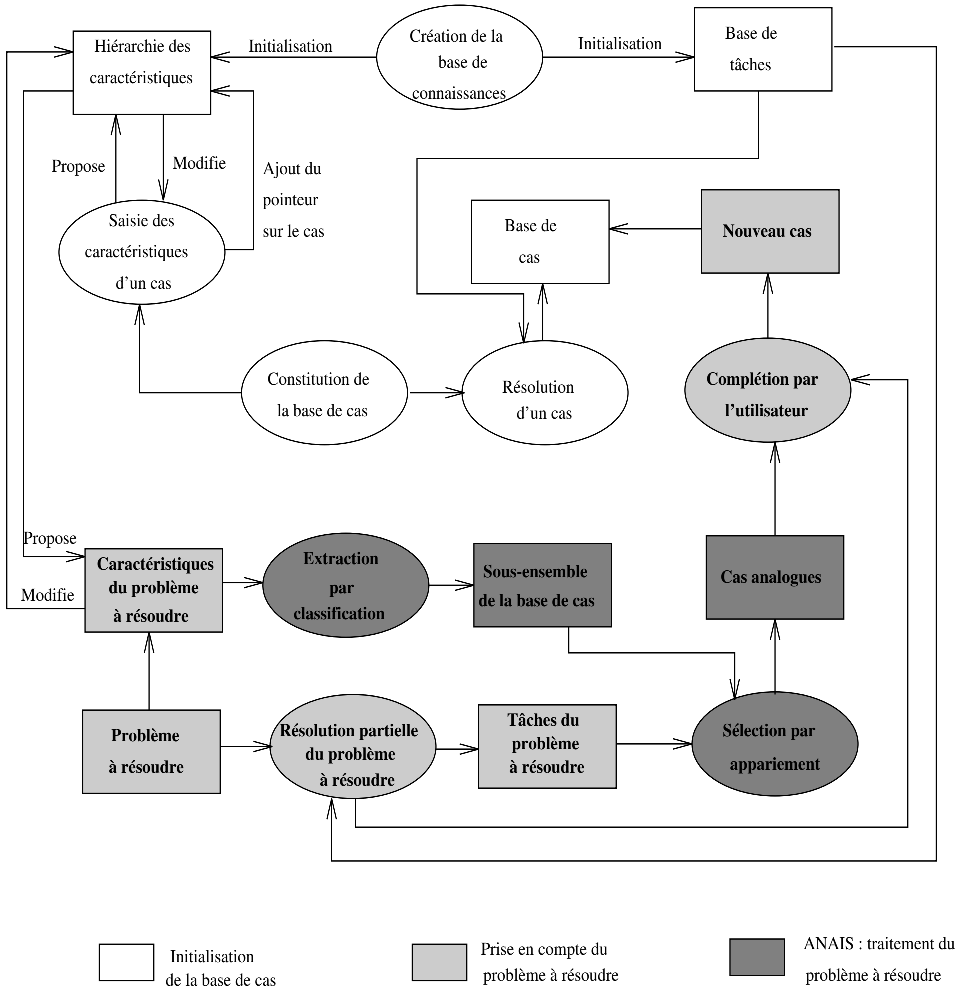


FIG. 3.1 - : Schéma fonctionnel d'ANAÏS

3.3.1 Initialisation de la base de cas

Cette initialisation comprend deux parties différentes relatives aux deux grandes étapes d'ANAÏS. ANAÏS commence par extraire les cas ayant des caractéristiques communes avec le nouveau problème, puis leur résolution partielle ou complète. Dans la base de cas initiale, deux informations sont donc nécessaires pour un cas : ses caractéristiques et sa résolution. Cette première étape consiste à résoudre des cas avec la base de tâches actuelle et à saisir les caractéristiques de ces cas. Il s'agit donc d'une étape d'acquisition des connaissances proprement dite.

Nous avons choisi d'indexer les caractéristiques dans une hiérarchie, et ce, afin d'utiliser l'algorithme de classification inhérent à l'environnement de résolution de problèmes utilisé. Dans un premier temps, la hiérarchie des caractéristiques est à définir, puis toutes les caractéristiques de chaque cas de la base sont saisies. La mémorisation de toutes les caractéristiques d'un cas se fait par indexation du cas dans la hiérarchie des caractéristiques (voir section 3.4).

Chaque cas résolu est mémorisé : toutes les étapes de raisonnement (sous forme d'enchaînement de tâches) sont stockées dans une base de connaissances indépendante des caractéristiques. Pour cette mémorisation, chaque cas est résolu avec la base de tâches existante, l'historique des tâches lancées est assimilé aux étapes de raisonnement.

3.3.2 Prise en compte du nouveau problème à résoudre

Cette seconde phase comprend aussi deux parties indépendantes : la saisie des caractéristiques du problème à résoudre et la saisie du raisonnement partiel de ce problème. C'est aussi une phase d'acquisition des connaissances, mais cette fois, des connaissances issues d'un problème non complètement résolu.

La saisie des caractéristiques du problème à résoudre se fait de la même manière que précédemment. Il faut tout d'abord mettre à jour la hiérarchie des caractéristiques en ajoutant éventuellement de nouvelles caractéristiques inhérentes au nouveau problème, puis dans un second temps, saisir les caractéristiques du nouveau problème en l'indexant dans la hiérarchie. Il est supposé que les caractéristiques dans un domaine particulier sont en nombre fini, aussi, au bout d'un certain nombre de résolutions de problèmes, la base de connaissances des caractéristiques n'évoluera plus.

L'ensemble de la résolution partielle du problème à résoudre est l'historique des tâches exécutées grâce à la base de tâches actuelle. Celle-ci est mémorisée indépendamment des caractéristiques.

3.3.3 Traitement du problème à résoudre

Ce traitement se fait par le système lui-même et s'effectue en deux étapes. Une première étape est effectuée par un algorithme de classification, il s'agit de la phase d'extraction où les caractéristiques du problème à résoudre sont comparées aux différentes caractéristiques existantes. Les cas partageant le plus de caractéristiques avec le problème à résoudre sont mémorisés dans un ensemble où la phase de sélection viendra puiser les cas à examiner. Cette étape est décrite dans la section 3.4.

La seconde étape, au cœur de l'originalité d'ANAÏS est l'étape de sélection. À partir du schéma de raisonnement partiel du problème à résoudre et des schémas de raisonnement des cas, il faut apparier les réseaux de tâches pour proposer les cas les plus proches du problème à résoudre. Les différentes parties de cette étape sont décrites en détail dans le chapitre 4.

3.3.4 Les étapes en perspectives

Dans la mesure où l'un des objectifs est d'améliorer la base de connaissances et où la base de cas, au fur et à mesure de son enrichissement risque de saturer la mémoire, il est nécessaire d'introduire une fonction « d'oubli». Il est envisagé dans les perspectives de notre travail, deux étapes supplémentaires :

- La généralisation de cas à partir d'un certain nombre de cas observés analogues. La construction d'un modèle générique inhérent à cette généralisation et le remplacement de l'ensemble des cas analogues par leur modèle générique.

- Mise à jour de la base des caractéristiques relativement au modèle générique construit. Il s'agit donc ici de remplacer l'indexation des cas analogues dans la hiérarchie des caractéristiques par une indexation sur le modèle générique.

3.4 Indexation et extraction des cas par les caractéristiques

Après l'initialisation de la base de cas et la prise en compte du problème à résoudre, nous décrivons la troisième partie qui est en réalité la partie intégrante d'ANAÏS : le traitement du problème à résoudre.

Ce traitement est divisé en deux parties : une extraction et une sélection. Nous décrivons ici la première partie, c'est-à-dire l'extraction de cas à l'aide des caractéristiques du problème à résoudre et de celles des cas déjà résolus. Cette première grande phase d'ANAÏS repose sur l'algorithme de classification de SHIRKA (décrit en section 3.4.1). Il s'agit de réduire le nombre de cas à examiner pour la phase suivante. Pour sélectionner les cas similaires dans la mémoire, il est impensable de considérer un algorithme qui évalue la similarité avec tous les cas de la mémoire, surtout que celle-ci augmente au fur et à mesure de la résolution de problèmes. D'où la nécessité de cette première phase qui doit présélectionner des cas avant d'apparier les raisonnements. La première difficulté rencontrée pour le modélisateur est la formulation des caractéristiques d'un problème.

La description principale d'un cas étant son raisonnement, ses caractéristiques ne doivent pas uniquement dépendre du domaine considéré, mais doivent aussi dépendre du raisonnement même. L'étude des différentes caractéristiques possibles dans chaque domaine est en dehors de notre projet. Les caractéristiques sont supposées être extraites par un processus classique d'acquisition des connaissances, automatique ou non. En électromyographie (voir chapitre 7, par exemple, quelques caractéristiques seulement sont connues, comme : maladie générale, diagnostic suspecté à confirmer, et quelques symptômes importants, comme par exemple, fourmis dans les doigts, trouble du sommeil...).

3.4.1 SHIRKA : un système de représentation par objet

Tous les types de connaissances, qu'il s'agisse de tâches ou de données, se définissent sous une seule entité SHIRKA appelée « schéma ». Un schéma définit en même temps une classe d'objets et un objet particulier. Il est décrit par une liste d'attributs et chaque attribut peut posséder des facettes qui peuvent être soit des attachements procéduraux qui calculent une valeur ou lancent un traitement particulier, soit des contraintes sur les valeurs de l'attribut (voir figure 3.2). Les valeurs d'un attribut sont aussi des objets et peuvent donc posséder des attributs, on parle alors d'objets complexes.

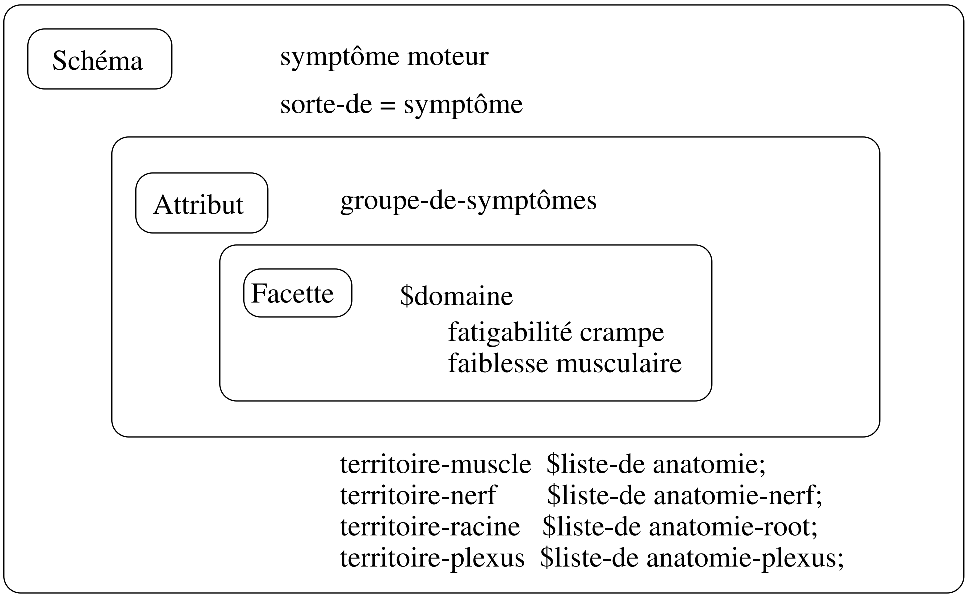


FIG. 3.2 - : Un schéma SHIRKA dans la base de connaissances électromyographique

Dans la figure 3.2, la classe (ou schéma) de base est : « symptôme moteur». C'est une sous-classe de la classe « symptôme» et elle possède cinq attributs. Ces attributs sont : *groupe-de-symptôme, territoire-muscle, territoire-nerf, territoire-racine* et *territoire-plexus*. Chacun de ces attributs a un type, c'est-à-dire une classe de rattachement. Par exemple le type de l'attribut *territoire-muscle* est *anatomie-muscle*. La classe *anatomie-muscle* possède 41 instances dont par exemple :

```
{ extensor-digitorum-communis
est-un = anatomie-muscle ;
region = foot ;
nerf-innervant = peroneus ;
site  = "distal" ;
racine-associee = L5 }

{ masseter
est-un = anatomie-muscle ;
region = face ;
nerf-innervant = facialis ;
site  = "facial" ;
racine-associee = without-root }
```

Un attribut *territoire-muscle* doit avoir comme valeur l'une de ces 41 instances. Chaque attribut peut posséder une« facette», dans le schéma « symptôme-moteur», par exemple, l'attribut *groupe-de-symptôme* possède la facette *domaine*. Cette dernière indique en fait le type de l'attribut qui est un type énuméré, c'est-à-dire qui doit avoir une ou plusieurs des trois valeurs suivantes : *fatigabilité, crampe, faiblesse musculaire*.

La classe proprement dite décrit l'ensemble des objets qui lui appartiennent. Un nouvel objet appelé instance est créé à partir du modèle de sa classe. Les classes sont organisées en hiérarchie, une classe fille hérite de tous les attributs de la classe mère, mais peut spécifier des contraintes plus fortes sur la valeur des attributs et, de plus, peut posséder des attributs supplémentaires. Le lien entre la classe mère et la classe fille est appelé « lien de spécialisation » et est représenté par l'attribut « sorte-de ». L'inférence des valeurs d'attributs peut se faire à l'aide des attachements procéduraux, par héritage, par défaut (facette valeur par défaut) ou par filtrage.

3.4.2 Hiérarchie des caractéristiques

Les caractéristiques des problèmes sont représentées à l'aide de schémas SHIRKA. Dans un fichier de données SHIRKA, c'est-à-dire comprenant classes, sous-classes et instances, la hiérarchie des caractéristiques est définie. Une hiérarchie indépendante décrit la connaissance de toutes les caractéristiques des problèmes stockés dans la mémoire. Toutes les caractéristiques existantes du domaine d'application sont supposées être stockées et classifiées dans la hiérarchie. Ajouter des nouvelles caractéristiques signifie modifier la hiérarchie en question, cela peut se traduire par le déplacement de certaines classes, la création d'un nouvel attribut ou encore la création d'une nouvelle classe.

Lors de la modification de cette hiérarchie, des problèmes de cohérence peuvent apparaître, lors par exemple du déplacement d'une classe, il faut vérifier l'héritage des attributs ainsi que les contraintes associées à ces attributs. De tels problèmes font l'objet d'autres recherches [Cap93].

Pour chaque classe de cette hiérarchie, les attributs correspondent aux caractéristiques, et un attribut spécial correspond à un pointeur sur la liste des problèmes ayant les caractéristiques de la classe où il sont rattachés (Figure 3.3). Un problème rattaché à une classe de cette hiérarchie possède toutes les caractéristiques de sa classe et hérite de celle des classes mères. Par exemple, en électromyographie, le cas *cas11* est une maladie musculaire dont le diagnostic reste à confirmer et qui possède des antécédents cliniques.

La description complète détaillant les étapes de raisonnement menant à la solution du *%cas17%* n'est pas comprise dans cette hiérarchie mais est stockée séparément dans la mémoire. L'extraction hiérarchique est faite par l'algorithme de classification de SHIRKA.

La classification peut prendre différentes significations en IA [Mar93]. Il peut tout d'abord s'agir de création de classes à partir d'un ensemble d'exemples, ce processus est aussi appelé catégorisation [CHL+91] et est surtout utilisé en apprentissage automatique (regroupement conceptuel) [MS83]. La seconde signification consiste, soit à trouver la classe d'appartenance d'une instance dans une hiérarchie, soit à ajouter une nouvelle classe dans la hiérarchie. Dans notre cas, il s'agit de trouver la classe d'appartenance d'une instance dans la hié-

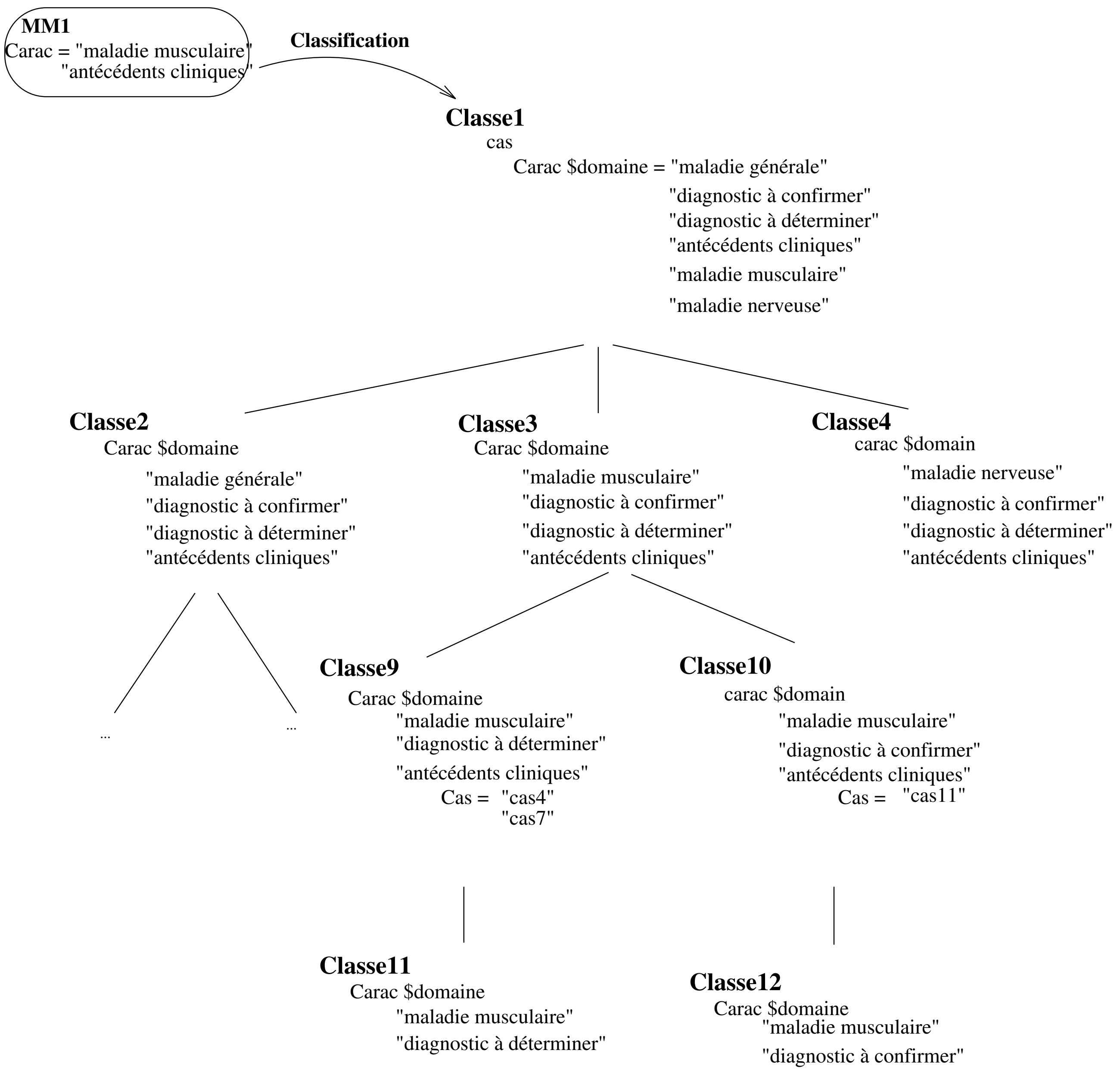


Fig. 3.3 - : La hiérarchie des caractéristiques dans la base de connaissances électromyographique

rarchie existante.

Rappelons que SHIRKA est un système de représentation de connaissances dont les éléments de base sont les classes et les instances (voir 4.1.1). Toute instance possède une classe de rattachement initiale et les valeurs des attributs de l'instance vérifient les contraintes (domaine, intervalle...) de la classe de création. Toutes les classes sont sous-classes de la classe objet et au pire, l'instance est rattachée à la classe objet. Trois propriétés différentes sont affectées à une classe vis-à-vis d'une instance à classifier : sûre, possible et impossible.

À une étape de classification, pour savoir si une classe est sûre, possible ou impossible pour une instance donnée, l'algorithme regarde si toutes les contraintes de chaque attribut de la classe sont vérifiées. Si toutes les valeurs des attributs de l'instance sont connues et respectent les contraintes, la classe est sûre. Si certaines valeurs d'attributs ne sont pas connues, elles sont, soit calculées (par attachement procédural, par la facette $valeur ou $defaut), soit demandées à l'utilisateur. Si ces valeurs manquantes ne sont finalement pas connues et que les autres valeurs connues des attributs vérifient les contraintes de la classe, la classe est possible. Dès qu'une contrainte d'un attribut de la classe est violée par la valeur d'un attribut de l'instance, la classe est impossible.

L'algorithme de classification de SHIRKA se fait en profondeur d'abord. De même que toute classe au dessus d'une classe sûre est une classe sûre, toute sous-classe d'une classe impossible est impossible. L'algorithme examine donc chaque classe et propage la propriété de la classe dans la hiérarchie. Toutes les branches de la hiérarchie sont explorées jusqu'aux nœuds terminaux (les classes les plus spécifiques).

En se replaçant dans notre contexte, lorsque l'utilisateur tente de résoudre un problème, il fournit quelques unes, si possible, de ses caractéristiques en les choisissant dans un domaine. Par exemple, supposons que pour notre problème à résoudre, il s'agisse d'une maladie musculaire (cas 1 : myopathie ou myosite). Une instance ***MM1*** de la classe la plus générale (classe 1) est créée avec comme valeurs de l'attribut « carac », la valeur « maladie musculaire » et « antécédents cliniques ». Dans la classe 1, qui est la classe la plus générale, le domaine des valeurs possibles pour l'attribut « carac » est l'ensemble de toutes les caractéristiques possibles. La valeur de l'attribut « cas » n'est pas connu car c'est cette valeur que l'on veut récupérer. La valeur de l'attribut

« similarité » n'est pas connu non plus car celle-ci est déterminée par l'algorithme d'appariement décrit dans la section suivante. Mais comme la classe 1 est la classe de création, cette classe est déterminée par l'algorithme de classification comme une classe sûre.

L'algorithme examine ensuite la classe 2. Les contraintes sur la valeur de l'attribut « carac » sont exprimées par la facette $domaine qui indique l'ensemble des valeurs possibles. Or dans la classe 2, le domaine de l'attribut « carac » contient la valeur « maladie générale ». Cette valeur n'étant pas présente dans les valeurs de l'attribut « carac » de l'instance, la classe 2 est considérée comme impossible. De même la classe 4 ayant comme valeur possible dans le domaine de l'attribut « carac » la valeur « maladie nerveuse », la classe est aussi considérée comme impossible. La classe 3 par contre ne possède aucune valeur incompatible avec celles de l'instance, elle est donc une classe possible. Elle ne peut être sûre car l'attribut « cas » n'est pas encore valué. Notons l'existence d'un attribut supplémentaire, l'attribut « similarités » qui sera renseigné par l'algorithme d'appariement de la seconde phase (voir section 6.3).

Les sous-classes de la classe 3 sont alors examinées. Les valeurs du domaine de l'attribut « carac » de la classe 9 sont : maladie musculaire, diagnostic à déterminer et antécédents cliniques. Aucune valeur de l'attribut « carac » de l'instance à classifier n'est différente d'une des valeurs du domaine de la classe 9, donc celle-ci est une classe possible. Il en est de même pour la classe 10. La classe 11 est une classe impossible car la valeur « antécédents cliniques » n'est pas dans le domaine de valeur de la classe 11. De même pour la classe 12.

La liste des classes sûres et possibles est donc la suivante : classe1, classe3, classe9 et classe10. À l'aide de cette liste de classes, une énumération des cas de cette liste de classes est possible grâce à la valeur de l'attribut « cas » de chaque classe de la liste. Le résultat final est l'ensemble des valeurs contenues dans l'attribut « cas » de chaque classe possible : *cas4, cas7, cas11*. Cette liste de cas partageant des caractéristiques avec le nouveau problème constitue un filtre dans la mémoire. Cette étape allège la suivante, qui constitue le principal problème de sélection des raisonnements similaires, aidant ainsi l'utilisateur à résoudre son problème.

Chapitre 4

ANAÏS : appariement de schémas de raisonnement

La phase d'extraction de problèmes par les caractéristiques étant effectuée, la sélection de problèmes similaires à un problème donné reste la plus complexe. Pour représenter les cas, sont mémorisés, en plus des caractéristiques, l'ensemble des étapes de résolution du problème.

Les entrées la phase de sélection décrite dans ce chapitre sont donc les suivantes :

- Les décompositions complètes des problèmes extraits
- La décomposition partielle du nouveau problème à résoudre

La sortie est le problème ou les problèmes les plus similaires, pour ce faire il faut déterminer une mesure de similarité.

Après avoir introduit la représentation des connaissances inhérente à l'environnement de résolution de problèmes utilisé, nous décrirons de manière détaillée l'algorithme utilisé dans ANAÏS.

Nous remarquons ensuite, que proposer le ou les problèmes les plus similaires dépend peu de l'algorithme mais repose essentiellement sur la définition d'une mesure de similarité. Cette notion de mesure de similarité est très importante en IA et le choix d'une telle mesure conditionne très souvent les résultats obtenus. Dans le chapitre suivant, nous examinons des mesures de similarité et leur utilisation en IA, avant de proposer notre propre mesure.

4.1 Représentation de la résolution d'un cas

4.1.1 SCAI : vers une modélisation générique du raisonnement

SCAI (Scientific Computing with Artificial Intelligence) est un environnement de résolution de problèmes développé au sein de l'équipe SHERPA [PR91]. Son objectif premier est de permettre le développement de systèmes à base de connaissances utilisant des ensembles de programmes ou de méthodes scientifiques déjà opérationnels. SCAI est défini dans le modèle objet SHIRKA [Rec85], ce qui permet de structurer les connaissances en hiérarchies de classes et instances. La démarche de résolution de problèmes est modélisée d'une part, à l'aide de tâches représentant les connaissances fonctionnelles nécessaires à la résolution, et, d'autre part, avec les procédures représentant les connaissances opératoires. Une base de connaissances est représentée par une base de tâches liées les unes aux autres par des liens séquentiels, de choix ou de spécialisation. Cette décomposition correspond à celle d'un problème en sous-problèmes, et ceci jusqu'à la définition des tâches élémentaires que sont les procédures. L'exploitation de la base de connaissances s'effectue par un « moteur » qui décompose récursivement la tâche en cours de traitement.

Représentation des connaissances

Il existe plusieurs types de connaissances : les tâches complexes (décomposées en sous-tâches), les méthodes (ou tâches terminales) et les données. Toutes les tâches (complexes ou terminales) comportent des entrées et des sorties typées, ces types correspondent aux données et sont des objets SHIRKA organisés en hiérarchie. Tous les types de connaissances qu'il s'agisse de tâches ou de données, se définissent sous la même entité SHIRKA appelée le schéma (voir figure 4.1).

Les données d'une application dans SCAI sont des schémas et peuvent être des classes et des instances. Les tâches sont des sous-classes des classes pré-définies de SCAI : tache-term, tache-seq, tache-choix-automatique et tache-choix-interactif (voir figure 4.1).

```
{tache
      sorte-de   =  schéma}

{tache-scai
      sorte-de   =     objet ;
      est-un     =     tache ;
      invisible  $un   booléen}

{tache-seq
    sorte-de   =  tache;
    ss-tache   $liste-de tache}

{tache-choix-auto
    sorte-de   =  tache;
    ss-tache   $liste-de tache}

{tache-term
    sorte-de   =  tache;
    ss-tache   $liste-de tache}
```

FIG. 4.1 - : La définition des tâches de SCAI dans le modèle SHIRKA.

Les attributs des tâches sont les entrées, les sorties et un attribut particulier qui énumère ses sous-tâches ou la procédure associée. La classe d'appartenance des attributs étant les classes pré-définies et les classes données du domaine. Les valeurs de ces attributs sont donc dans les instances données pour l'application.

```
{res-equa
    sorte-de = tache-scai;
    est-un = tache-seq;
    &entree a $un entier;
    &entree b $un entier;
    &entree c $un entier;
    &sortie r1 $un reel;
    &sortie r2 $un reel;
    ss-tache = { sorte-de = calc-det;
                     c1 $var<- a;
                     c2 $var<- b;
                     c3 $var<- c;
                     det $var-nom de }
                 { sorte-de = calc-rac;
                     det1 $var<- de;
                     b11 $var<- b;
                     a11 $var<- a;
                     rac11 $var-> r1;
                     rac21 $var-> r2 }}

{calc-rac
    sorte-de = tache-scai;
    est-un = tache-choix-auto;
    &entree det1 $un entier
     $var-nom de;
    &entree b11 $un entier;
    &entree a11 $un entier;
    &sortie rac11 $un reel;
    &sortie rac21 $un reel;
    cs-tache = { sorte-de = calc-rac-pos;
     det $var<- de
     $a-verifier
                     {positif det $var<-de};
                     b1 $var<- b11;
                     a1 $var<- a11;
                     rac1 $var-> rac11;
                     rac2 $var-> rac21}
                 {sorte-de = calc-rac-neg;
                     rac1 $var-> rac11;
                     rac2 $var-> rac21 }}
```

FIG. 4.2 - : Les attributs des tâches : entrées, sorties et sous-tâches.

Dans la figure 4.2, les attributs « d'entrée » de la tâche *res-equa* sont les suivants : a représentant la valeur du coefficient de second degré de x, b la valeur du coefficient de premier degré de x et c la constante de l'équation ($ax^2 + bx + c = 0$). Les sorties sont le résultat du calcul des deux racines de l'équation. Dans l'attribut « ss-tache», sont définies les sous-tâches de *res-equa* qui seront effectuées en séquence puisque *res-equa* est une « tache-seq». La première sous-tâche *calc-det* est une tâche terminale calculant le déterminant de l'équation. Elle prend comme entrée les trois coefficients de l'équation (appelés c1, c2 e c3 dans cette tâche) et renvoie en sortie la valeur du déterminant dans l'attribut « det». Ce dernier attribut est passé dans la sous-tâche suivante *calc-rac* par l'intermédiaire de la facette « var-nom». Cette facette permet de repérer l'attribut « det» par le nom « de». Les entrées de la seconde sous-tâche (tâche terminale) sont : det1 (qui prend la valeur renvoyée dans « de»), a et b, coefficients de équation. Les sorties sont les résultats des calculs des racines passés dans les attributs « rac1» et « rac2». La valeur des sorties de la seconde sous-tâche sont alors passées dans les attributs « r1» et « r2» de la tâche principale *res-equa*.

La tâche *calc-rac* est une tache de choix automatique. Ses sous-tâches sont *calc-rac-pos*et *calc-rac-neg*. La première sous-tâche n'est lancée que si l'attachement procédural *positif* lié à l'attribut d'entrée *det* est vérifié, c'est-à-dire si le déterminant de l'équation du second degré est positif. Dans le cas contraire, c'est la seconde sous-tâche *calc-rac-neg* qui est lancée. Cette dernière remplit les deux racines avec la valeur nulle indicative car les deux racines n'existent pas.

Les tâches complexes. Elles représentent la décomposition du problème en sous-problèmes et permettent de modéliser l'ensemble des chemins possibles pour résoudre un problème. La résolution s'exprime donc à travers la description de l'ensemble des tâches choisies parmi tous les enchaînements possibles. Il existe trois types de tâches complexes :

- Les tâches dites « séquentielles » (tache-seq dans le modèle), c'est-à-dire dont les sous-tâches s'effectuent séquentiellement. Les entrées d'une sous-tâche peuvent utiliser les sorties de la sous-tâche précédente ou les entrées de la tâche mère. Les sorties d'une sous-tâche sont données soit par les autres sous-tâches de celle-ci si la sous-tâche est complexe, soit par calcul dans une procédure appelée par la sous-tâche si celle-ci est terminale. Les sorties de la tâche mère sont donc remplies par les sous-tâches qui la composent.

– Les tâches dites « de choix » (tache-choix-auto ou tache-choix-interactif dans le modèle), c'est-à-dire dont on choisira la sous-tâche appropriée. Ces tâches permettent donc plusieurs résolutions différentes. Si c'est l'utilisateur qui choisit lui-même la sous-tâche à effectuer, on parlera de tâche de choix interactif. Si c'est le système qui détermine lui-même la sous-tâche à réaliser, on parlera de tâche de choix automatique. Ce dernier cas est le plus souvent utilisé car le choix dépend généralement du contexte de résolution.

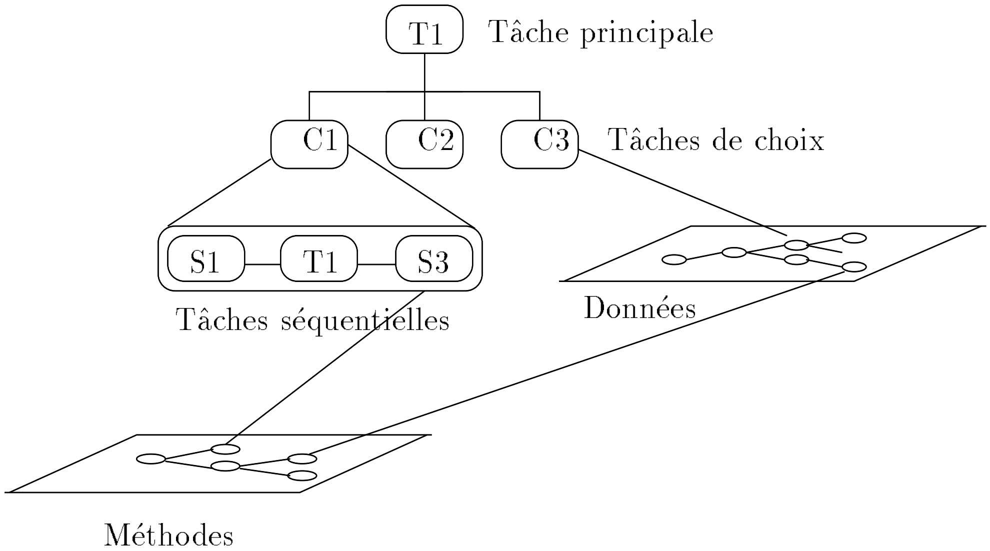


FIG. 4.3 - : Les types de connaissances dans SCAI : les tâches décrivent la résolution du problème, les méthodes décrivent des inférences élémentaires et des données décrivent les concepts du domaine.

Dans la figure 4.3, la tâche complexe t1 est une tâche de choix, possédant trois alternatives de tâches : c1, c2 et c3. Les deux dernières tâches sont terminales, tandis que c1 est aussi une tâche complexe. Cette dernière est une tâche séquentielle qui se décompose dans l'enchaînement des trois tâches : s1 (tâche terminale), t1 (la tâche principale) et s3 (tâche terminale). Le rappel de la tâche t1 dans s1 est une récursion donc indique que t1 est une tâche itérative. Une tâche complexe permet ainsi de représenter l'itération. Pour la tâche en cours de résolution, il suffit que l'une de ses sous-tâches soit une tâche de plus haut niveau dans la hiérarchie des tâches.

Les tâches terminales. Elle ne sont pas décomposables et sont rattachées à des méthodes. Une méthode est comme une tâche complexe, une classe ayant des attributs d'entrée et de sortie et un attribut spécial précisant le nom de la procédure utilisée. Les méthodes sont organisées en hiérarchie et le lien de spécialisation d'une classe à une sous-classe signifie une restriction sur les conditions d'exécution. Il est possible de rattacher plusieurs procédures à une méthode. À une procédure est rattachée une fonction LISP qui est directement exécutable (voir figure 4.4).

Dans la figure 4.4, l'attribut appelé *proc* de la tâche terminale *calc-det* (qui calcule la valeur du déterminant de l'équation du second degré $c_1x^2 + c_2x + c_3$) détermine le nom de la procédure utilisée. Les trois entrées *c1 c2* et *c3* sont passées dans les trois entrées *c11 c21* et *c31* de *proc-calc-det*. De même la sortie *det1* de la procédure *proc-calc-det* est passée dans la sortie *det* de la tâche *calc-det*. L'attribut nommé *nom-fct* de la procédure précise le nom de la fonction LISP utilisée. Enfin, la dernière chaîne du maillon est la fonction LISP *calc-det*. Cette dernière récupère les attributs d'entrée *c11 c21 c31* de la procédure *proc-calc-det* avec la fonction *val?*, calcule la valeur du déterminant, puis passe cette valeur à l'attribut de sortie *det1* de *proc-calc-det* par la fonction d'affectation *affect*.

Dans cet exemple la méthode n'est pas spécialisée. Mais dans des bases de connaissances de calcul scientifique plus élaborées, une méthode peut posséder plusieurs sous-classes (des sous-méthodes), permettant l'application d'une procédure adaptée au contexte. Ainsi, dans l'exemple de [PR91], la méthode d'inversion d'une matrice possède deux sous-méthodes : la méthode de Gauss dans le cas général et la méthode

```
{calc-det
    sorte-de = tache-scai;
    est-un tache-term;
    &entrée c1 $un entier;
    &entrée c2 $un entier;
    &entrée c3 $un entier;
    &sortie det $un reel;
    proc = {sorte-de = proc-calc-det;
              c11 $var <- c1 ;
              c21 $var <- c2 ;
              c31 $var <- c3 ;
              det1 $var -> det } }
```

La tâche terminale

```
{proc-calc-det
    sorte-de = procedure;
    nom-fct $valeur calc-det;
    &entrée c11 $un entier;
    &entrée c21 $un entier;
    &entrée c31 $un entier;
    &sortie det1 $un réel }
```

La procédure

```
(de calc-det (inst)
    (let ((a (val? 'c11))
          (b (val? 'c21))
          (c (val? 'c31)))
        (setq det1 (-(* b b)(* 4 a c)))
        (affect 'det1 det1)))
```

La fonction LISP

FIG. 4.4 - : Une tâche terminale SCAI calculant la valeur du déterminant dans une équation du second degré

de Cholesky dans le cas où la matrice est symétrique définie positive (voir figure 4.5).

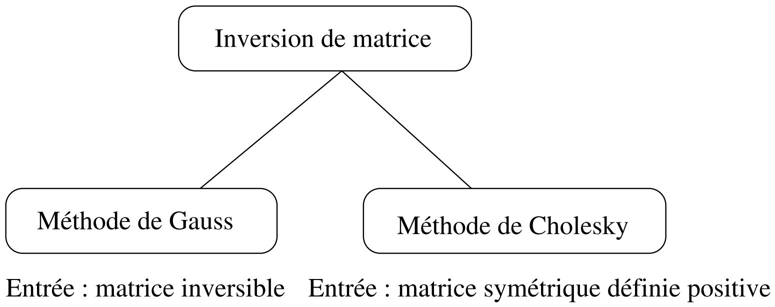


FIG. 4.5 - : La hiérarchie des méthodes associée à l'inversion de matrice

Les données. Les données sont représentées par des classes SHIRKA organisées en hiérarchie. Certaines classes sont pré-définies comme les classes *entier* ou *réel*. Dans chaque application, les données du domaine sont nécessaires et les classes définies peuvent être très nombreuses. En électromyographie, notamment, plus de 300 classes sont définies ainsi qu'une soixantaine d'instances [ZVC92] (voir figure 4.6).

```
{hypothèse-de-maladie
        sorte-de = objet;
            système-suspect $un chaine;
            atteinte $un chaine;
            régions $liste-de region;
            protocole-associé $un protocole}

{région
        sorte-de = objet;
            composants $liste-de region;
            l-nerfs $liste-de anatomie-nerf;
            l-muscles $liste-de anatomie-muscle}

{main
        sorte-de = région;
            composants = doigtI doigtII doigtIII doigtIV doigtV;
            l-muscles = CAP CA5 }
```

FIG. 4.6 - : Des données en électromyographie

Le mécanisme d'exploitation des connaissances : le moteur de SCAI. Le moteur de SCAI utilise en grande partie le mécanisme de classification de SHIRKA, d'une part pour spécialiser les tâches terminales et d'autre part, pour retrouver des valeurs d'attributs dans la hiérarchie des données. Nous laisserons de côté l'utilisation de la classification pour décrire plus particulièrement le mécanisme d'exploitation des tâches complexes (décomposition et passage de paramètres). Lors de la phase de résolution de problème, l'utilisateur choisit une tâche à résoudre qui constituera la tâche principale. L'utilisateur crée une instance de cette tâche principale et donne des valeurs aux attributs d'entrée.

Deux phases sont alors alternées dans SCAI :

- La phase de spécialisation où le moteur va chercher à classifier l'instance et propose à l'utilisateur les sous-tâches admissibles.
- La phase de décomposition : le moteur recherche l'ensemble des sous-tâches et traite différemment les tâches séquentielles, de choix ou terminales.
  - Dans le cas d'une tâche séquentielle, le moteur crée une instance de la première sous-tâche en lui passant les attributs d'entrée nécessaires, puis il lance la résolution de cette nouvelle sous-tâche. Si la première sous-tâche réussit, le moteur passe à la seconde sous-tâche en passant les attributs de sortie de la sous-tâche vers la tâche principale. Si la résolution de la première sous-tâche échoue, la tâche principale passe en échec et la résolution du problème a échoué.
  - Dans le cas d'une tâche de choix, une seule des sous-tâches doit réussir pour que la tâche réussisse. Si le choix est automatique, le moteur crée une instance de la première sous-tâche, si elle réussit, les sorties de la sous-tâche sont passées dans la tâche principale et la résolution de celle-ci s'arrête. Si la première sous-tâche échoue, le moteur crée une instance de la seconde sous-tâche. Dès que l'une des sous-tâches réussit, la résolution s'arrête. Si le choix est interactif, la résolution est identique, sauf que l'utilisateur choisit l'ordre des sous-tâches à effectuer.

- Dans le cas d'une tâche terminale, une instance de la méthode est créée, le moteur essaie ensuite de spécialiser cette méthode dans la hiérarchie des méthodes, puis crée une instance de la procédure associée à la méthode la plus spécifique. Si la procédure réussit, la tâche terminale réussit du même coup.

La démarche du moteur se fait donc récursivement et toujours en commençant par spécialiser puis en décomposant la tâche en cours de traitement.

Deux modes de fonctionnement sont actuellement disponibles dans SCAI, le mode libre et le mode guidé. Dans le mode guidé, l'utilisateur donne une instance de la tâche de plus haut niveau à résoudre et le moteur effectue les enchaînements des sous-tâches selon la décomposition. Dans le mode libre, l'utilisateur effectue lui-même les tâches dans l'ordre où il le désire. Cela peut être intéressant pour un utilisateur qui expérimente un nouvel enchaînement de méthodes ou qui essaie de résoudre un problème hors des limites de ceux résolvables par la base de connaissance actuelle. Ceci peut se produire lorsque la base de connaissances est incomplète ou lorsqu'on la construit justement par résolution de nouveaux problèmes.

Lorsque la base de connaissances est complexe, il est de plus en difficile à l'utilisateur de choisir les enchaînements de tâches quand celle-ci sont très nombreuses. L'utilisateur peut être perdu dans son raisonnement et pour répondre à ce problème, une étude a été faite [NS91], permettant d'aider l'utilisateur dans la phase de résolution.

4.1.2 Aide à la résolution de problèmes : la reconnaissance de plans.

Suite à l'introduction du mode libre dans SCAI, une aide à l'utilisateur a été nécessaire. L'utilisateur aura lancé un certain nombre de tâches dans un ordre correspondant à la résolution de son problème. Si à un moment de la résolution, l'utilisateur est perdu, il peut alors faire appel à l'algorithme de reconnaissance de plan. Cet algorithme se base sur l'historique des tâches lancées par l'utilisateur. On appelle « plan », l'ensemble des instances de tâches résolvant un problème particulier.

Le plan de l'utilisateur est constitué d'un ensemble d'actions (instances de tâches). Ces actions sont du type : décomposer le problème en sous-problèmes, choisir de résoudre un problème en utilisant telle ou telle méthode. L'environnement de résolution de problèmes ne garde de ces actions que l'historique des tâches dans l'ordre où elles ont été lancées. À partir de cet historique, l'algorithme essaie de deviner la stratégie employée (c'est-à-dire les choix effectués) et ne peut la retrouver que si elle est contenue explicitement dans la base. L'algorithme indique alors à l'utilisateur les tâches qu'il pourrait lancer pour terminer la résolution de son problème.

Pour indiquer à l'utilisateur les tâches complémentaires qu'il aurait dû exécuter, l'algorithme compare la stratégie incomplète de l'utilisateur avec tous les plans de résolution de la base de connaissances. Ces plans sont en fait toutes les instanciations possibles des tâches dans tous les chemins existants dans la base. Pour constituer l'ensemble de ces chemins, toute l'arborescence de la base de tâche est parcouru en passant par tous les choix (tâches de choix interactifs où intervient l'utilisateur) possibles.

Pour réaliser la correspondance entre le plan de l'utilisateur et les plans de la base, l'algorithme descend dans l'arbre de résolution de la base (grâce aux liens de spécialisation et de décomposition) en créant d'abord une instance de la tâche de plus haut niveau. Le moteur est ensuite lancé, ce dernier crée des instances des tâches de décomposition, jusqu'à ce qu'il trouve une tâche lancée par l'utilisateur. Dès qu'une tâche lancée par l'utilisateur est trouvée, SCAI la résout complètement et compare cette résolution effectuée dans la base de connaissances avec celle de l'utilisateur (tâche principale identique et sous-tâches de cette tâche lancée à la main par l'utilisateur). Les tâches manquantes sont alors indiquées mais la fin de la résolution n'est pas effectuée, elle reste à la charge de l'utilisateur.

Dans la figure 4.7 les cercles représentent les méthodes et les rectangles les tâches. Lorsque les méthodes ont un sous-arbre associé, ce sont des méthodes à spécialiser ou complexes, sinon elles sont directement attachées à une procédure. Les objets noircis (tâches et méthodes) sont ceux que l'utilisateur a effectivement lancés, tandis que les autres sont ceux présents dans la base de tâches. Les objets grisés sont les tâches ou méthodes que l'utilisateur n'a pas lancés et qui auraient dû

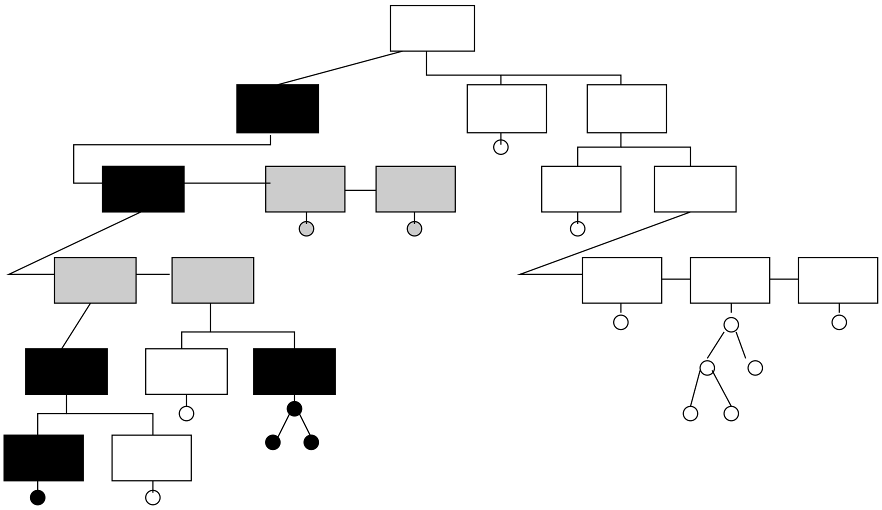


FIG. 4.7 - : Historiques des tâches lancées par l'utilisateur

l'être pour résoudre complètement le problème. Ce sont ces objets qui sont proposés comme aide à l'utilisateur lorsqu'il fait appel à la reconnaissance de plans.

L'algorithme de reconnaissance de plans compare donc l'ensemble des tâches lancées par l'utilisateur avec tous les réseaux générés à partir de la base (Figure 4.7). Si la stratégie choisie n'existe pas dans la base sous la forme d'un réseau exactement équivalent, l'algorithme indique que sa stratégie n'existe pas. L'utilisateur aura peut-être tenté de résoudre un problème « nouveau » par rapport à la base de connaissances et aura utilisé des portions de stratégie résolvant d'autres types de problèmes dans le même domaine.

Cet algorithme examine de manière exhaustive l'ensemble des plans de résolution d'un problème contenus dans la base. Cette recherche est très coûteuse en temps car les bases de tâches opérationnelles sont importantes et l'espace de recherche peut donc être très grand. L'exploration de tout cet espace est contraire aux principes généralement utilisés en apprentissage symbolique automatique où des heuristiques sont utilisées pour réduire cet espace. En ASA, ce choix a priori est surtout motivé par des raisons de complexité. D'autre part l'algorithme de reconnaissance de plans ne répond pas au problème de la recherche d'une résolution de problèmes similaires.

4.2 Sélection des cas par appariement de schémas de raisonnement dans ANAÏS

4.2.1 Description générale

La phase de sélection a pour but de déterminer le problème de la base de cas le plus similaire au problème en cours de résolution. Dans un premier temps, ANAÏS déterminera l'ensemble des tâches communes entre les cas de la base extraits de la première phase et le problème à résoudre. Le problème à résoudre est exprimé par un ensemble de tâches décomposées par l'environnement de résolution de problèmes, ces tâches étant jugées utiles mais incomplètes pour terminer la résolution du problème en cours (voir figures 4.8). L'ensemble de ces tâches est noté RP1, RP2 et RP3 dans la figure.

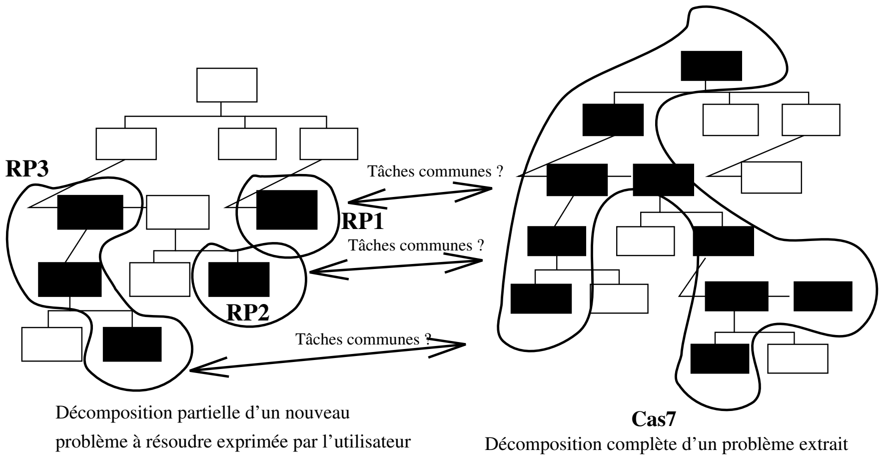


FIG. 4.8 - : Sélection du problème le plus similaire : recherche de tâches communes

Pour ce faire chaque partie de l'ensemble des tâches du problème à résoudre (chaque RPi) est comparé à chaque cas extraits de la base de cas, de cette comparaison ressortira une liste de tâches en commun (voir figure 4.9). De ces tâches en commun, une similarité est déterminée et un calcul global de la similarité entre le cas extrait et le problème à résoudre est calculé (voir chapitre 5).

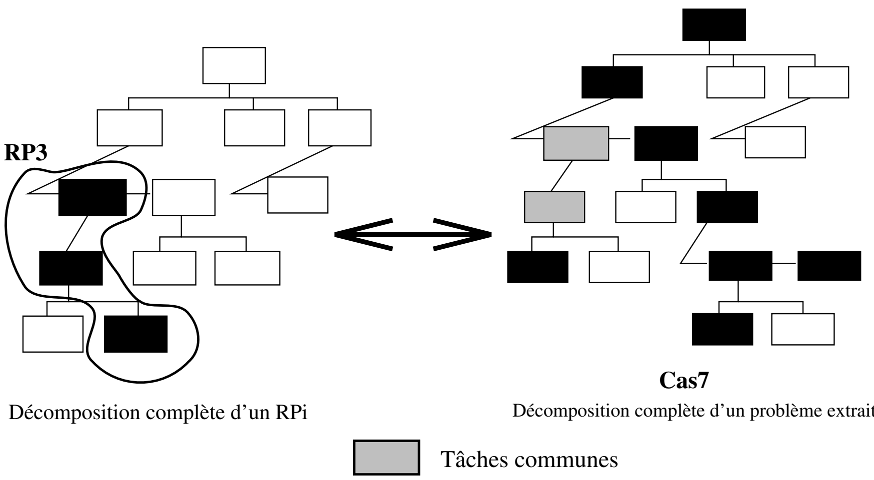


FIG. 4.9 - : Sélection du problème le plus similaire : recherche de tâches communes

4.2.2 Description sur un exemple

Une base de tâches test a été créée de manière à utiliser pleinement les tâches de choix automatiques ou interactifs, de séquence ou itératives. La base de tâches utilisée pour illustrer le comportement d'ANAÏS a été élaborée (voir figure 4.10). Dans les bases existantes, il m'existe en effet que très peu de tâches de choix-interactif (une ou deux dans IFREMER et pas du tout dans les autres bases, par exemple dans la base BIOMOL [MWS+93] qui traite de la biologie moléculaire. Ceci ne laisse à l'utilisateur que la liberté de choisir des tâches terminales, alors que différents enchaînements choisis pour une même tâche constituent de nombreuses libertés supplémentaires qui amènent autant de possibilités différentes de raisonnements.

Cette base vise à résoudre l'équation $F(x) = 0$, où F est un polynôme, par différentes méthodes. Selon le degré de l'équation, une méthode plus ou moins complexe est utilisée. Lorsque le degré est zéro, nous considérons qu'il n'y a pas de solutions. Lorsque le degré vaut 1, la valeur sera directement calculée ($ax = b \Rightarrow x = -\frac{b}{a}$). Lorsque le degré vaut 2, l'algorithme classique est utilisé (calcul du déterminant et des racines si le déterminant est positif). Enfin, lorsque le degré est supérieur à 2, trois méthodes au choix peuvent être exécutées :

1. La méthode de Newton. Celle-ci revient à substituer à la courbe représentative de $y = F(x)$, sa tangente au point X_i considéré. La dérivée est approximée par la formule de Taylor jusqu'au degré 1 : $F(x+h) = F(x) + hF^1(x)$ Un point initial X_0 est choisi, $F(X_0)$ est calculé puis une suite de points X_i est construite telle que $X_{i+1} = \frac{F(X_i+h)-F(X_i)}{h}$. La valeur recherchée est obtenue lorsque $\mid X_{i+1} - X_i \mid < \varepsilon$, où ε est une précision donnée par l'utilisateur. La difficulté avec cette méthode est qu'il n'est pas certain qu'elle converge. Pour remédier à cette difficulté, un nombre maximal d'itérations est permis (choisi par l'utilisateur).

2. La méthode par dichotomie. On part de deux valeurs X_1 et X_2 telles que $F(X_1).F(X_2) < 0$. Au moins une racine X_0 est donc comprise entre X_1 et X_2. Une suite de points X_i est ensuite calculée, telle que $F(X_i).F(X_{i+1}) < 0$ soit toujours négatif. Une des valeurs X_1 ou X_2 est remplacée par le milieu de X_1 et X_2, de telle manière que $F(X_1).F(X_2)$ soit toujours négatif. La valeur recherchée est obtenue lorsque $\mid X_1 - X_2 \mid < \varepsilon$, où ε est une précision donnée par l'utilisateur. Cette méthode est forcément convergente si la fonction est continue. La seule difficulté est que l'utilisateur doit connaître deux valeurs initiales x0 et x1 telles que $f(x0).f(x1) < 0$.

3. La méthode des approximations successives. L'équation $F(X) = 0$ est remplacée par l'équation $F(X) + X = X$. Le point fixe de $F(X) + X$ est ensuite calculé en construisant une suite $X_{i+1} = G(X_i)$ où G(x) = F(x) + x, jusqu'à une précision ε donnée. Cette méthode n'est pas forcément convergente car il est possible qu'il n'y ait aucune solution. Un nombre d'itérations maximal donné par l'utilisateur est donc nécessaire pour assurer la fin du processus de calcul.

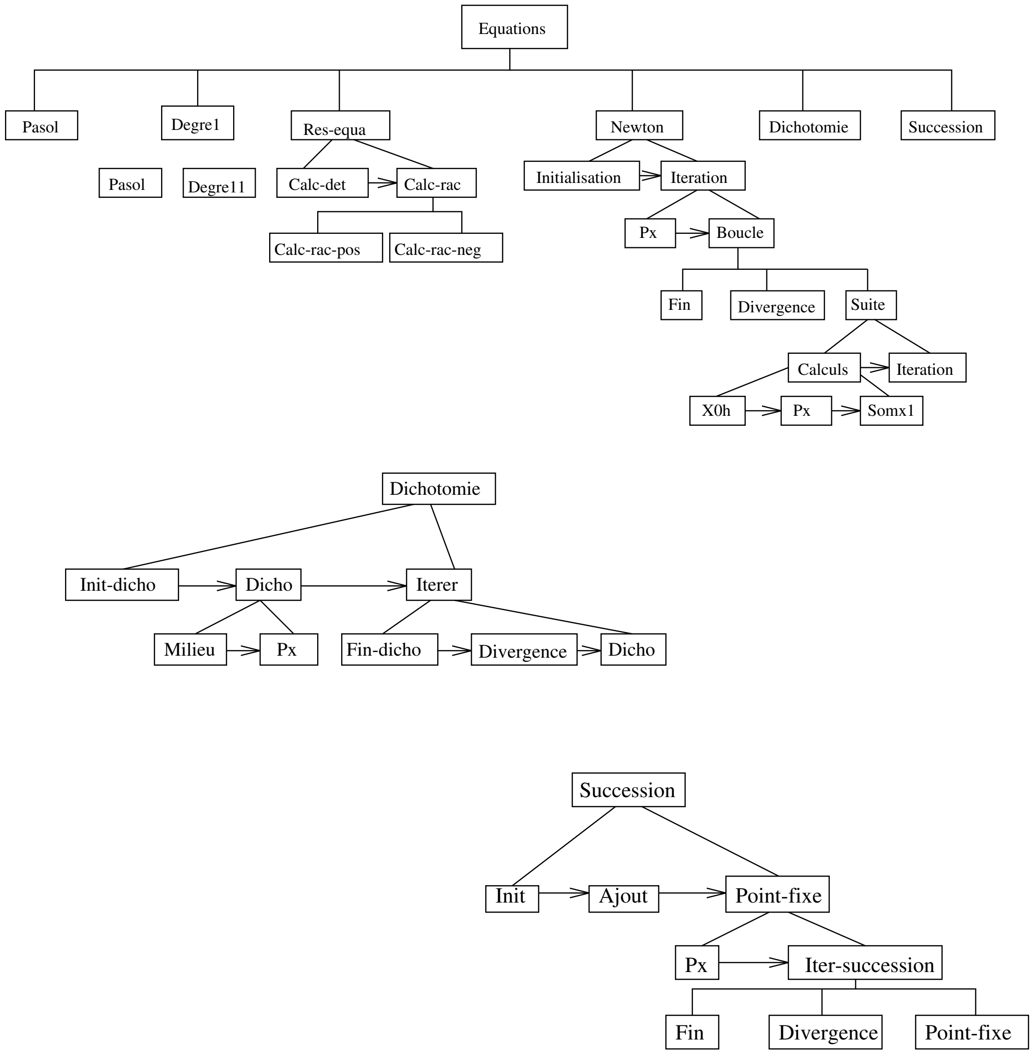


FIG. 4.10 - : la base de tâches F(x)=0

Après le chargement de cette base de tâches, plusieurs problèmes sont résolus pour constituer la base de cas. Les différentes tâches de ces problèmes sont décrites à l'aide d'une liste en notation post-fixée. Ainsi, par exemple le cas1 est noté :

```
(cas1 (newton   (initialisation
  ((iteration
     (px boucle
(suite
   (calculs
      (x0h px somx1))))) 17)
    iteration (px boucle (divergence)))))
```

Chaque nouvelle parenthèse ouverte est une sous-tâche directe de la tâche avant la parenthèse et chaque parenthèse fermée indique que l'ensemble des sous-tâches termine la tâche en cours de résolution. Les appels successifs de la tâche itération sont remplacés par le nombre de fois où *iteration* est appelée. Le cas1 correspond à la résolution de équation $F(x) = 0$ par la méthode de Newton.

Supposons les autres cas suivants dans la base :

```
(cas2 (dichotomie (init init-dicho
                   ((dicho (milieu px) iterer) 19)
                   iterer (fin-dicho))))
```

et

```
(cas3 (succession
         (init ajout
             ((point-fixe (px iterer-succession)) 13)
       point-fixe (fin))))
```

Ces deux derniers cas correspondent à la résolution de $F(x) = 0$ par la méthode de dichotomie et par la méthode des approximations successives. L'utilisateur essaie ensuite de résoudre un nouveau problème et exprime des tâches utiles :

```
(RP1 (x0h px somx1)).
(RP2 (dicho (milieu px))).
(RP3 (px)).
```

L'algorithme d'appariement des réseaux a pour but de donner le cas le plus proche du raisonnement partiel exprimé par les tâches utiles au problème à résoudre (les RPi). Pour cela, il faut avoir une mesure de similarité, des importances relatives pour les tâches et connaître l'ensemble des tâches partagées par un cas et les RPi. Avant de déterminer l'ensemble des tâches partagées, certaines règles de simplification, inhérentes à l'environnement de résolution de problèmes SCAI, doivent être appliquées.

4.2.3 Deux règles de simplification

La première règle de simplification concerne les tâches récursives. Dans la base de tests par exemple la tâche *iteration* est récursive et appelée un nombre de fois indéterminé à priori.
Supposons pour simplifier l'écriture que le cas4 utilisant la méthode de Newton appelle deux fois la tâche *iteration* avant de donner la solution finale. Ce dernier s'écrit alors :

```
(cas4 (newton (initialisation
                iteration
                    (px boucle
                         (suite
                            (calculs (x0h px somx1)
                               iteration
                                 (px boucle
                                       (suite
                                         (calculs (x0h px somx1)
                                            iteration
                                              (px boucle (fin))))))))))
```

Nous remplacerons les appels successifs de *iteration* par le nombre de fois où la tâche *iteration* est appelée. Le dernier appel de *iteration* est laissé tel quel car il ne possède pas la même sous-tâche.

```
(cas4 (newton (initialisation
                ((iteration
                    (px boucle
                      (suite
                        (calculs (x0h px somx1))))) 2)
               iteration
                 (px boucle (fin)))))
```

Dans cet exemple de base de connaissances, il peut ne pas être important de connaître le nombre d'appels successifs d'une tâche mais il se peut que dans d'autres bases, celui-ci soit important. Au chapitre 7, nous verrons que dans le domaine électromyographique, il est effectivement important de connaître le nombre d'appels de la tâche controle-protocole. Le raisonnement global du cas électromyographique dépend fortement de ce nombre d'appels.

Une seconde règle de simplification est nécessaire. En effet, lorsqu'une tâche terminale est exécutée, la méthode associée est automatiquement lancée, et cette dernière est aussi définie par une tâche SCAI. Lorsque l'on retrouve l'historique des tâches lancées, on retrouve donc pour chaque tâche terminale la méthode correspondante.

Dans les exemples précédents, la règle a déjà été appliquée, par exemple : (RP1 (x0h (somme) px (proc-px) somx1 (procx1)) a été remplacé par : (RP1 (x0h px somx1)) où *somme proc-px* et *procx1* sont respectivement les méthodes associées aux tâches terminales *x0h*, *px* et *somx1*.

4.2.4 Recherche des tâches partagées

Dans l'algorithme d'appariement, une première étape, après l'application des règles ci-dessus, est de rechercher les tâches partagées entre les schémas de raisonnement partiels exprimés par l'utilisateur et les schémas de raisonnement nécessaires à la résolution des cas en mémoire. Les entrées de l'algorithme de recherche de tâches partagées sont d'une part, les listes de tâches réalisant les cas (un arbre avec une seule racine par cas) et d'autre part, des listes de tâches utiles au problème à résoudre (un seul arbre aussi avec une seule racine). La sortie est une liste contenant l'ensemble des tâches communes pour chaque cas. L'algorithme est le suivant :

1. **Pour** chaque cas à examiner **faire**
2. **Pour** chaque RPi **faire**
   (a) Mettre la tâche principale du RPi dans *tacheRPi*.
   (b) Mettre le pointeur *Ptachecas* sur les tâches du cas à : 1 (on part de la première tâche contenue dans le cas).
   (c) Mettre la liste des tâches partagées *listecommune* à vide.
   (d) **Tant que** on n'est pas à la dernière tâche du cas **et** *listecommune* n'est pas maximale **faire**
      i. Recherche(tacheRPi,Ptachecas,liste)
      ii. **Si** longueur(liste) > longueur(listecommune) **alors** affecter liste dans listecommune.
   (e) **Fin Tant que**
3. **Fin pour** tous les RPi
4. **Fin pour** tous les cas à examiner

Dans la procédure *recherche*, il s'agit de rechercher la plus longue liste de tâches communes en balayant les tâches du cas à partir de l'indice Ptachecas. La procédure *recherche* a comme entrées *tacheRPi*, la tâche principale de RPi, *Ptachecas*, le pointeur indiquant l'endroit où on se trouve dans le cas. La sortie de la procédure *recherche* est *liste* qui représente la liste des tâches communes entre le RPi et le cas à partir de Ptachecas dans le cas. Cette procédure est la suivante :

Recherche(tacheRPi,Ptachecas,liste)

1. Initialiser liste à vide
2. **Tant que** non trouvé(tacheRPi) et non fin(cas) **Faire**
   Incrémenter Ptachecas
3. **Fin tant que**
4. **Si** trouvé(tacheRPi) **Alors**
   (Regarder les tâches suivantes du RPi)
   (a) Mettre pointeurRPi à 1
   (b) **Tant que** non fin(RPi) et tachecas = tacheRPi **Faire**
      i. Incrémenter pointeurRPi
      ii. Incrémenter Ptachecas
      iii. Concaténer *tachecas* à la liste des tâches communes *liste*
   (c) **Fin Tant que**
5. Fin Si

Soit n, le nombre de tâche du cas et m, le nombre de tâches du RPi. Lors de la première recherche de la liste des tâches communes dans le cas, on part de la première tâche du cas, puis on balaye toutes les tâches du cas jusqu'à trouver (ou ne pas trouver) la tâche première du RPi. Cette recherche comporte donc au maximum n comparaisons. Puis si la tâche première du RPi est trouvée à un endroit P du cas, on regarde si les sous-tâches suivantes du RPi sont aussi dans le cas, et ce, à partir de P. Ces comparaisons sont au nombre maximum de m (nombre de tâches du RPi). Ce qui donne au total, pour cette première recherche n+m comparaisons.

Mais il se peut que la liste des tâches communes ne soit pas maximale et que l'on puisse en trouver une plus grande dans le cas. Si toutes les tâches du cas n'ont pas été balayées, on continue à rechercher une autre liste de tâches communes éventuellement plus grande.

Dans la seconde recherche, le nombre de tâches du cas à partir desquelles se fait la recherche est réduit car on ne recherche plus dans les tâches du cas déjà examinées. Dans le pire des cas, la recherche est effectuée à partir de la tâche suivante du cas, donc on effectue n-1 comparaisons. De même la recherche suivante exige au pire n-2 comparaisons. Le coût maximal obtenu est donc : (n+m)+((n-1)+m)+((n-2)+m)+...+(1+m) soit nm + n(n-1)/2. C'est un coût polynomial équivalent à la recherche d'un sous-arbre maximum dans un arbre.

Ne sont appariés avec les RPi, que les cas extraits dans la première phase. Le nombre de cas total à examiner est donc réduit, ce qui diminue le temps de calcul global. Il se peut cependant qu'aucune des caractéristiques ne soit donnée au départ, tous les cas de la base de cas sont alors examinés et la recherche peut s'avérer assez longue. Pour pallier cette difficulté, il est envisagé dans l'évolution future d'ANAÏS, de généraliser des cas similaires (à partir d'un certain nombre déterminé) et de remplacer l'ensemble de ces cas similaires par leur généralisation. Ce qui permettrait de diminuer le nombre de cas et donc forcément le nombre de cas à apparier si les caractéristiques du nouveau problème n'étaient pas définies.

La recherche des tâches communes entre un RPi et un cas est appelée appariement simple. Pour les exemples précédents cela donne les résultats suivants :

- Entre cas1 et RP1 : (x0h px somx1).
- Entre cas1 et RP2 : ().
- Entre cas1 et RP3 : (px).
- Entre cas2 et RP1 : ().
- Entre cas2 et RP2 : (dicho (milieu px)).

- Entre cas2 et RP3 : (px).
- Entre cas3 et RP1 : ().
- Entre cas3 et RP2 : ().
- Entre cas3 et RP3 : (px).

La détermination de l'ensemble des tâches communes au raisonnement partiel du problème à résoudre et aux cas extraits est la première étape du calcul proprement dit de la similarité. Cette similarité est d'ailleurs plus une mesure de qualité qu'une similarité pure. La similarité totale dépend surtout du résultat de la recherche de tâches communes, c'est une similarité structurelle. Avant de présenter la similarité utilisée dans ANAÏS, différents types de similarité sont présentées et discutées dans le chapitre 5 et la mesure de similarité utilisée est justifiée.

4.3 Complétion par l'utilisateur et mise en mémoire

Une fois les cas sélectionnés par appariement de raisonnement, les cas les plus similaires au problème à résoudre sont connus. Mais que faire de ces cas analogues ? Le problème à résoudre apparaît à l'utilisateur sous forme graphique de tâches et sous-tâches, et proposer les cas analogues dans la même représentation possède des avantages certains. D'une part le raisonnement complet d'un cas apparaît à l'écran et d'autre part, l'utilisateur peut voir du premier coup d'œil les différences de tâches entre le cas et le problème. Bien que l'interface graphique visualisant les cas analogues n'ait pas été réalisée, il est envisagé plus qu'une visualisation de ceux-ci. Une manière conviviale pour l'utilisateur étant en effet de pouvoir venir piocher des tâches d'un cas analogue en les « copiant » puis en les « collant » dans son problème à résoudre.

En créant interactivement des nouvelles tâches et en utilisant les tâches des cas analogues, l'utilisateur peut créer son propre schéma de raisonnement pour le problème à résoudre. Lors de la création de nouvelles tâches dans la base de tâches, des vérifications sont nécessaires pour assurer la cohérence de la base. Notamment, lorsqu'une tâche nouvelle est créée sous-tâche d'une autre tâche, il faut s'assurer que ses entrées sont incluses dans la tâche de plus haut niveau. Si ce n'est pas le cas, il faut ajouter les entrées correspondantes dans la tâche existante. De même, pour les sorties de la nouvelle tâche, il faut s'assurer qu'elles appartiennent bien à la tâche de la base, sinon, il faudra venir les ajouter. D'autre part, il doit être impossible de créer une sous-tâche d'une tâche terminale à moins de modifier le type de celle-ci, ce qui entraîne la disparition des procédures associées.

Après création des nouvelles tâches et ajout de tâches dans le problème à résoudre, ce dernier est mieux décomposé mais pas nécessairement complet. Le cycle de recherche de cas analogues peut alors être relancé, ce qui peut affiner les cas analogues recherchés. Lorsque le problème à résoudre est enfin complet, si l'utilisateur juge qu'il est nécessaire de le mémoriser, le nouveau problème est stocké dans la base de cas. Cette mémorisation se fait d'une part, en ajoutant le problème dans la hiérarchie des caractéristiques grâce aux caractéristiques données au départ et d'autre part, en mémorisant le schéma complet de raisonnement.

Les opérations à effectuer après obtention des cas analogues ne semblent donc pas poser de difficultés théoriques. La difficulté est de fournir les cas analogues et surtout de sélectionner les meilleurs parmi ceux extraits dans la première phase. La mesure de similarité employée est donc cruciale et une étude des mesures de similarité utilisées dans le domaine et dans les domaines adjacents au RBC est donc nécessaire afin de définir une mesure propre à ANAÏS qui soit adéquate.

Chapitre 5

Similarité : un problème crucial en intelligence artificielle

La notion de similarité au sens le plus général du terme est utilisée dans plusieurs domaines. Elle est utilisée dans le classement des objets et dans la formation de concept. Elle est aussi étudiée dans le domaine de la psychologie. Nous observons deux types de similarité : la similarité numérique et la similarité symbolique. Par exemple, une distance mathématique est une similarité numérique et elle est définie par les propriétés qu'elle doit vérifier. Dans le domaine de la vision et notamment de la reconnaissance des formes, une similarité numérique est utilisée, une image étant modélisée par des nombres proportionnels aux différents niveaux de gris. Reconnaître une forme dans une image correspond à apparier une partie de l'image avec une forme pré-définie.

Dans le domaine de l'IA, la similarité est surtout utilisée en apprentissage symbolique où l'on essaie de déterminer une ressemblance entre individus. Deux types de mesures de similarités sont reconnues en apprentissage automatique et en RBC [Car86, Mig92]. Une mesure utilisant des connaissances de surface (caractéristiques généralement liées au domaine) et une mesure dite structurelle utilisant des connaissances liées à la structure d'un exemple, problème ou cas, la connaissance utilisée étant appelée connaissance profonde [Bis93, Gen83]. Une similarité n'est pas forcément entièrement de surface ou structurelle, elle peut être plus ou moins structurelle et de surface, son calcul est rendu d'autant plus complexe qu'elle est structurelle. Dans la mesure où ANAÏS travaille essentiellement sur les données symboliques (les tâches), plusieurs similarités symboliques sont présentées.

5.1 Similarités en analyse de données

L'analyse de données permet de traiter des données en grand nombre. L'analyse peut être exploratoire (classification par exemple) ou confirmatoire (prise de décision et visualisation). L'analyse de données est une collection de méthodes permettant de modéliser des informations et de trouver des relations entre objets ou entre les paramètres caractérisant ces objets.

Nous nous intéressons à la partie de l'analyse de données qui étudie la notion de ressemblance entre individus. Celle-ci est utilisée notamment lors de la classification automatique par partition, arbre ou hiérarchie. Les mesures de ressemblance issues de l'analyse de données ont fortement inspiré les mesures de similarité utilisées en apprentissage symbolique automatique. Nous supposerons que les données sont homogènes, c'est-à-dire que toutes les variables des individus sont de même nature.

5.1.1 Définitions

Est appelée mesure de ressemblance [DLPT82] toute application à valeurs numériques qui permet d'exprimer numériquement un lien existant entre les individus ou entre les variables. Rappelons que l'on appelle « individu» toute donnée représentée à l'aide de plusieurs paramètres (appelés variables).

Un indice de similarité sur un ensemble Ω d'individus est une application S qui vérifie les trois propriétés suivantes :

1. S est une application Ω^2 dans R^+.
2. S est symétrique : $\forall(w_1, w_2) \in \Omega^2 : S(w_1, w_2) = S(w_2, w_1)$.
3. $\forall(w_1, w_2) \in \Omega^2$ avec $w_1 \neq w_2 : S(w_1, w_1) = S(w_2, w_2) > S(w_1, w_2)$

La dernière condition indique que la similarité entre un élément et lui-même est maximale et a toujours la même valeur pour tous les individus de la population Ω.

Un indice de dissimilarité est une application $\tilde{S}$ qui satisfait aux deux premières conditions de l'indice de similarité (application et symétrie) et en plus à la condition suivante : $\forall w \in \Omega$, $\tilde{S}(w, w) = 0$. Ce qui signifie simplement que la dissimilarité entre un individu et lui-même est nulle. L'indice de dissimilarité peut se définir à l'aide de l'indice de similarité par la relation suivante : $\tilde{S}(w_1, w_2) = S(w_1, w_1) - S(w_1, w_2)$.

Une distance euclidienne est un indice de dissimilarité qui vérifie en plus les propriétés suivantes :

1. $s(w_1, w_2) = 0 \Longleftrightarrow w_1 = w_2$
2. $\forall (w_1, w_2, w_3) \in \Omega^3, s(w_1, w_2) \leq s(w_1, w_3) + s(w_3, w_2)$

Une distance ultramétrique est un indice de dissimilarité vérifiant les propriétés :

1. $s(w_1, w_2) = 0 \Longleftrightarrow w_1 = w_2$
2. $\forall (w_1, w_2, w_3) \in \Omega^3, s(w_1, w_2) \leq Max(s(w_1, w_3), s(w_3, w_2))$

5.1.2 Différentes distances

La distance de Minkowski [DLPT82], englobe quant à elle plusieurs distances, puisqu'elle dépend d'un paramètre λ :

$$d(w_1, w_2) = [\sum_{j=1}^{m} \mid x_{i_1}^j - x_{i_2}^j \mid^{\frac{1}{\lambda}}]^{\lambda}$$

Cette distance est dite semi-euclidienne car pour certaines valeurs de λ, elle n'est plus quadratique. Lorsque λ vaut 2, on retrouve la distance euclidienne.

La distance du χ^2 [DLPT82] est importante et souvent utilisée. Soient x_i et $x_{i'}$ deux individus ayant p variables x_i^j et $x_{i'}^j$ où j est le numéro de la variable. Pour chacune de ces variables, il existe un certain nombre de valeurs possibles dans l'ensemble de tous les individus.

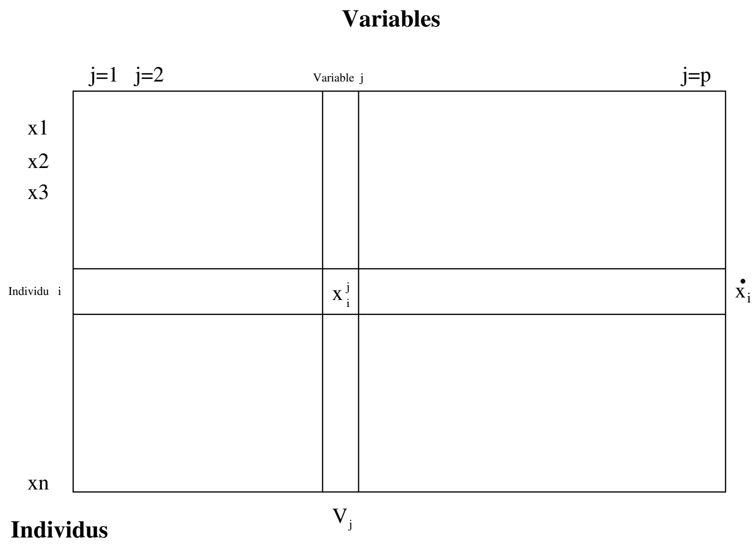


FIG. 5.1 - : Représentation des individus en analyse de données

Soit $V_j = \sum_{i=1}^{n} x_i^j$ et $\dot{x}_i = \sum_{j=1}^{p} x_i^j$
À l'aide de ces deux sommes, la distance du χ^2 s'écrit :

$$d^2(x_i, x_i') = \sum_{j=1}^{p} \frac{1}{V_j} \left(\frac{x_i^j}{\dot{x}_i} - \frac{x_{i'}^j}{\dot{x}_i'} \right)^2$$

Cette distance prend donc en compte les différentes valeurs possibles de chaque variable et en cela, elle est bien adaptée aux tableaux de fréquences.

Il existe des distances adaptées lorsque deux individus sont à valeurs booléennes ou binaires, elles sont basées sur la fréquence d'apparition des deux valeurs dans l'un et l'autre des individus.

À partir de distances entre individus, il est possible de définir des mesures de ressemblance entre groupes. Étant donné deux groupes d'individus A_1 et A_2 et une mesure de ressemblance, il existe plusieurs possibilités pour calculer la ressemblance entre deux groupes qui peuvent être par exemple, le lien minimum, maximum ou de manière plus complexe la distance des centres de gravité ou distance des noyaux.

5.1.3 Conclusion des mesures utilisées en analyse de données

Les mesures de ressemblance sont nombreuses en analyse de données et il n'est pas toujours facile de choisir la mesure appropriée aux besoins de l'utilisateur. Il existe cependant certains critères à partir desquels le choix d'une mesure est plus ou moins adéquate. Notamment lorsque l'utilisateur a une idée de la proximité de certains individus, il peut comparer son évaluation par rapport aux mesures proposées.

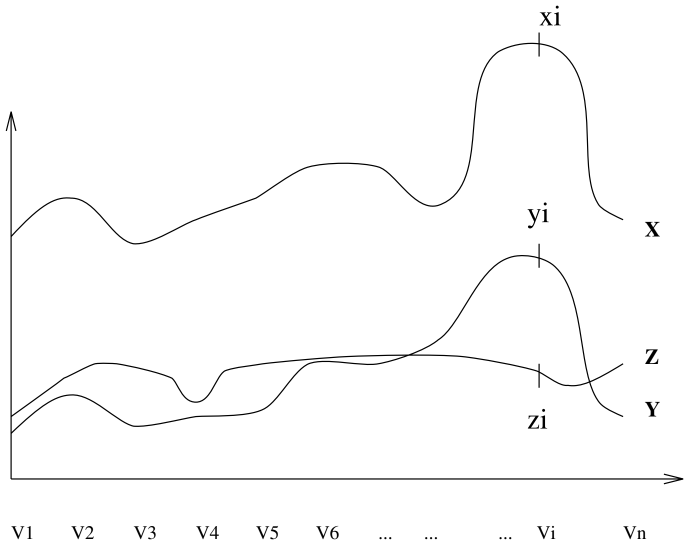


FIG. 5.2 - : Choix d'une distance. X,Y et Z représentent les individus, V_1, V_2 ...V_n, les variables des individus et l'axe vertical est celui des valeurs des variables. Dans cet exemple, les variables prennent toutes leurs valeurs dans le même ensemble et les données X, Y et Z sont homogènes.

Certaines mesures, comme la distance du χ^2, privilégient la ressemblance des profils des individus. Dans la figure 5.2, $d(x, y) < d(x, z)$ car le profil de x est plus proche de celui de y que de z. La différence entre chaque x_i et y_i doit être presque toujours la même. Deux individus ont exactement le même profil si la différence entre les valeurs des individus est constante pour chaque variable. Ce qui se traduit par la formule suivante: $\mid x_i - y_i \mid = Cst, \forall i$. Le profil est donc d'autant plus proche que la valeur $\sum_{i=1}^{n} \mid x_i - y_i \mid - \mid x_j - y_j \mid$ est petite.

D'autres mesures privilégient surtout la distance entre les valeurs moyennes des valeurs de deux individus (exemple la distance de Minkowski pour $\lambda = 1$). Deux individus x et y sont d'autant plus proches que $\sum_{i=1}^{n} \mid x_i - y_i \mid$ est petite. Dans le choix d'une telle distance, $d(x, y) > d(x, z)$ dans la figure 5.2.

En analyse de données, les données doivent suivre un format bien déterminé en se représentant sous forme d'individus possédant des valeurs pour un certain nombre de paramètres. Cette représentation s'avère particulièrement adaptée à la représentation par attributs-valeurs en apprentissage symbolique.

Dans ce domaine, la classification automatique se propose non seulement de déterminer les catégories, mais aussi de calculer la similarité entre les instances de manière à minimiser la similarité inter-catégories et à maximiser la similarité intra-catégorie. Dans ce contexte, le choix d'une mesure de similarité équivalente à celles de l'analyse de données peut s'avérer insuffisante. Un des buts de la classification automatique est de déterminer des nouvelles mesures de similarité satisfaisant aux formats des données et aux contraintes inter et intra-catégories énoncées ci-dessus.

Dans le domaine limitrophe de l'IA, qu'est la psychologie cognitive, différents modes de calcul de similarité correspondent à différents modes de pensée des êtres humains. Il est intéressant de se pencher sur ces méthodes de calcul, d'autant plus que l'acquisition des connaissances s'est, dans certains cas, fortement inspirée de la psychologie cognitive pour l'élaboration de ses modèles.

5.2 Similarités en psychologie cognitive

En psychologie cognitive, il existe différents modèles de processus d'évaluation. Chacun présente une manière différente de traiter la connaissance des experts en situation de prise de décision. Les modèles présentés sont issus de [Pin87].

1. Le modèle conjonctif.
   L'idée de base est que pour que l'entité soit jugée acceptable, tous les critères doivent être supérieurs à un seuil minimum appelé seuil

de satisfaction. Le jugement général porté sur un sous-domaine sera égal au minimum des jugements élémentaires.

2. Le modèle disjonctif.
   Il est aussi appelé « fonction maximum d'évaluation» car l'entité est jugée sur son meilleur aspect sans considération des autres attributs. Le concept de meilleur est représenté sous la forme d'un seuil. Ce modèle signifie qu'un dossier est considéré comme acceptable s'il est excellent sur au moins une caractéristique pertinente, quelque soit la valeur des autres. De manière générale, il s'agit de privilégier un attribut, celui considéré comme étant le plus important. La notion d'importance d'une caractéristique apparaît donc ici, sous sa forme la plus élémentaire.

3. Le modèle lexicographique.
   Il suppose que les caractéristiques sont observées par ordre d'importance. La caractéristique la plus importante est d'abord examinée. Si certaines conditions sont remplies (par exemple, la valeur d'un attribut est jugée suffisamment proche : un certain seuil de proximité est atteint), alors le processus d'évaluation est arrêté, sinon la caractéristique suivante dans l'ordre d'importance est examinée. Cette méthode est plus complète que la précédente dans la mesure où il est possible que tous les attributs soient pris en compte avec leur importance associée. Cependant, cela nécessite de définir un seuil et un ordre, soit arbitrairement, soit défini par l'utilisateur. S'il est effectivement défini par l'utilisateur, c'est une connaissance supplémentaire à acquérir ainsi que les importances de chaque attribut (si ceux-ci sont définis par le même utilisateur).

4. Le modèle compensatoire.
   Il compense de mauvais scores obtenus sur certaines caractéristiques par de bons scores sur d'autres attributs. Il évalue une entité en lui donnant une note globale selon une fonction d'évaluation pondérée. L'importance de chaque attribut constitue un indice de pondération et l'entité est évaluée en fonction de l'ensemble des attributs.
   Si N est le nombre d'attributs du domaine, S_j la similarité pour

deux exemples par rapport à l'attribut j, P_j le poids (ou importance) associé à l'attribut j et enfin S la similarité globale entre les deux exemples, alors il y a plusieurs manières de calculer la similarité dont par exemple :

$$S = \frac{1}{N}\sum_{i=1}^{N} P_j S_j$$

ou

$$\log(S) = \frac{1}{N}\sum_{i=1}^{N} P_j S_j$$

Ces méthodes de calcul prennent en compte les « ressemblances» de valeur entre attributs (les S_j) et les importances de tous les attributs (les P_j) vis-à-vis de l'ensemble des attributs. Les deux calculs correspondent à une mesure de similarité équivalente qui semble être une mesure de similarité plus adéquate dans la sélection du meilleur.

Ces différentes méthodes de calcul sont-elles utilisées dans la sélection de cas dans le RBC ? Ou sont-elles inadéquates ou incomplètes ? Afin de répondre à ces questions, différentes similarités en RBC et en ASA (Apprentissage Symbolique Automatique) sont examinées.

5.3 Similarités en apprentissage symbolique automatique

5.3.1 Similarités en « conceptual clustering »

Le « conceptual clustering » est une technique d'apprentissage automatique visant à regrouper des observations ou des données. L'objectif peut être soit de les regrouper afin de pouvoir induire la description d'un concept [Mic90], soit pour la construction automatique de taxinomies. Dans les deux cas, il est nécessaire de définir une mesure de similarité pour classer les nouveaux exemples dans les différents groupes existants. Dans CLUSTER/2 [Mic93], les connaissances sont représentées dans le langage APC qui est une extension du calcul des prédicats du premier ordre. Un exemple dans ce système se présente donc sous la forme d'une conjonction de prédicats :

e1 : vol(cou-tendu) $\wedge$ forme-bec(cassé) $\wedge$ couleur-bec(noir) $\wedge$ taille-cm(88).

Dans cet exemple, tiré de [Cer89], on cherche à classifier des oiseaux de la famille des échassiers. Les attributs (ou descripteurs) sont : vol, forme-bec, couleur-bec et taille-cm. Les valeurs de ces attributs sont : cou-tendu, cassé, noir et 88. Le choix d'un autre exemple pour produire une description regroupant les deux exemples se fait grâce à une fonction d'évaluation lexicographique.

Cette fonction est en quelque sorte une similarité puisque c'est grâce à elle que l'on va choisir le « meilleur exemple ». Il s'agit en fait de spécifier pour chaque exemple un critère de tolérance correspondant à un nombre minimum d'attributs communs. Il s'agit donc d'une similarité logique, où, de manière générale, la similarité est calculée en faisant le rapport entre le nombre d'attributs ayant la même valeur et le nombre total d'attributs. En prenant un second exemple :
e2 : vol(cou-tendu) $\wedge$ forme-bec(cassé) $\wedge$ couleur-bec(rose) $\wedge$ taille-cm(85)
La similarité logique la plus simple est : $S(e1, e2) = \frac{2}{4} = 0.5$. En réalité dans CLUSTER, il ne s'agit pas de choisir le meilleur exemple mais un exemple suffisamment proche, le critère de tolérance pouvant être défini par l'utilisateur en spécifiant notamment le nombre d'attributs minimal ayant la même valeur.

À partir de ce type de similarité, il est possible d'en dériver de plus complexes, prenant en compte par exemple, la distance entre deux valeurs d'attributs ou encore le poids de chaque attribut, ce qui semble plus proche de la réalité pour la sélection du « meilleur» [Tve77]. Ce type de similarité sera discuté dans les sections suivantes.

5.3.2 COBWEB : un exemple en formation de concepts

Description

La formation de concepts est une partie de l'apprentissage symbolique. Elle repose sur l'hypothèse que l'humain, à partir d'un ensemble d'observations, acquiert des nouveaux concepts et les organise en hiérarchies. Cette approche est analogue au « conceptual clustering», dans les deux cas le nombre de classes n'est pas connu *a priori* et aucune intervention extérieure permet de classer un objet. La différence essentielle entre le regroupement conceptuel et la formation de concepts est que l'apprentissage doit être incrémental. L'incrémentalité est ici une contrainte forte, car il ne suffit pas d'ajouter des objets dans la hiérarchie mais de reprendre en compte toutes les objets déjà mémorisés et de reconstruire la hiérarchie. [SF86]. L'approche observée est intermédiaire entre le « conceptual clustering » et le RBC. Ce dernier est en effet fondé sur la formation de l'expérience et est aussi incrémental.

Dans cette approche, une utilisation des hiérarchies est de classer des instances. Une instance est d'abord attachée au nœud le plus général (le plus haut) et descend dans la hiérarchie en modifiant si nécessaire cette hiérarchie. Lorsque l'instance est attachée à un nœud, donc à un concept, la description de ce concept apporte des connaissances supplémentaires sur l'instance. Dans cette approche de classification, l'analogie avec l'algorithme de classification de SHIRKA est très intéressante. La différence essentielle est que dans SHIRKA, une instance est classée dans la hiérarchie et peut n'appartenir à aucune des classes existantes. En formation de concept, il est fréquent qu'aucune classe définie ne puisse contenir une instance, la hiérarchie est alors modifiée de manière à intégrer l'instance.

COBWEB est un système de formation de concepts [Fis87] qui utilise une mesure de qualité des partitions issue de la psychologie cognitive. Par rapport aux mesures définies précédemment, celle de COBWEB est beaucoup plus complexe. Dans COBWEB, les concepts sont un ensemble de couples attribut/valeur. Deux informations sont utilisées pour choisir le concept à associer à une instance. La première est appelée prédictivité et correspond à la probabilité conditionnelle qu'une instance ayant la valeur v pour un attribut donné a, appartienne au concept c : $P(c|a=v)$. Cette probabilité est relative à l'apparition des concepts parents et est fonction du nombre d'instances déjà classées. La seconde information est la typicité d'un concept. Celle-ci est définie comme la probabilité conditionnelle qu'une instance appartenant à une classe c ait la valeur v pour l'attribut a : $P(a = v|c)$.

La construction de la hiérarchie se fait de manière incrémentale. Avec la première instance, une première classe est créée, puis avec la seconde instance deux classes filles sont créées, l'une contenant la première instance et l'autre la seconde. Lors de la prise en compte des instances suivantes, une fonction d'évaluation permet de choisir la branche à explorer (une nouvelle classe est alors éventuellement créée). Lorsqu'une instance est classée, les valeurs de prédictivité et de typicité sont modifiées.

Après chaque classification d'instances, deux opérateurs sont lancés pour optimiser la hiérarchie obtenue : un opérateur de fusion (*merge*) qui permet de créer une classe mère entre plusieurs classes et un opérateur de scission (*split*) qui détruit une classe mère inutile ([GLF89]).

Évaluation

Pour choisir la branche suivante où l'instance va descendre, il est défini une fonction d'évaluation qui dépend de tous les concepts possibles à un point donné de l'algorithme de classification. La fonction d'évaluation est issue de travaux en psychologie [GC85] et est le résultat de deux domaines : la théorie de l'information et la théorie des jeux. Cette fonction est définie de façon à maximiser les connaissances dérivées de l'appartenance d'une instance à un concept. La fonction dépend des deux informations mémorisées au fur et à mesure de la construction de la hiérarchie : la prédictivité et la typicité.

Pour une instance i donnée, d'attribut A_i et de valeur V_{ij} et pour une classe C_k, la prédictivité $P(A_i = V_{ij}|C_k)$ est calculée ainsi que la typicité $P(C_k|A_i = V_{ij})$. La fonction d'évaluation, appelée aussi mesure de qualité du regroupement conceptuel est définie dans un premier temps par :

$$\sum_i \sum_j \sum_k P(A_i = V_{ij}) P(C_k|A_i = V_{ij}) P(A_i = V_{ij}|C_k)$$

Où $P(A_i = V_{ij})$ est la probabilité que l'attribut A_i soit égal à la valeur V_{ij}. Ainsi la classe d'appartenance choisie est celle qui maximise l'utilité de la classe. À chaque fois qu'un choix se présente au niveau de la classe où l'instance doit migrer, cette fonction d'utilité est calculée pour chacune des classes en question. Si K est le nombre de classes possibles à un moment donné, l'expression finale de l'utilité (après application des règles de Bayes) d'une partition est la suivante :

$$\frac{\sum_i \sum_j \sum_{k=1}^{K} P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(a_i = V_{ij})^2}{K}$$

Cette mesure permet de maximiser la similarité des instances du même concept, représentée par la typicité et de maximiser la dissimilarité inter-classes, représentée par la prédictivité. Cela correspond tout à fait aux buts du *conceptual clustering*. D'autre part, la mesure d'évaluation maximise l'information déduite de l'appartenance d'une instance à un concept. Cela est tout à fait en harmonie avec les attentes des théories de l'information en psychologie cognitive.

5.3.3 KBG

KBG [Bis93] est un système d'apprentissage permettant entre autres, la classification conceptuelle des données. À partir d'un ensemble d'exemples, un module d'évaluation des similarités entre exemples précède le module de classification proprement dit. KBG s'appuie très fortement sur cette similarité dans tous les algorithmes utilisés. La mesure en question dépend très fortement du langage de représentation utilisé et est inspirée de l'analyse de données.

Langage de représentation

Il repose sur un sous-ensemble de la logique du premier ordre sans négation, symbole fonctionnel ni quantificateur existentiel. La structure de base du langage est l'atome qui est un prédicat d'arité quelconque (éventuellement nulle). Les prédicats représentent les propriétés et relations entre deux types de données : les entités et les valeurs. Exemples : *Taille(pierre, grand)*. Le prédicat *Taille* est binaire et exprime une relation entre la valeur *pierre* et la valeur *grand*. *Distance(a,b,petit)* : il existe deux entités *a* et *b* liées par la relation *distance* qui a pour valeur *petit*.

L'ensemble d'apprentissage est constitué d'exemples et de contre-exemples et d'une théorie du domaine (type de données utilisées, prédicats et règles). À chaque prédicat, il est possible d'associer un poids (valeur réelle positive) rendant compte de la valeur informative du prédicat. Par exemple: pour *couleur-de(entité,couleur)*, poids=1 et pour *distance(entité,entité,mètre)*, poids = 2,5.

Mesure de similarité

La mesure de similarité dans KBG est utilisée à la fois pour classifier et pour généraliser les exemples. Cette mesure permet de comparer deux valeurs quelconques d'un type donné et rend une valeur comprise dans l'intervalle [0..1], la valeur 1 exprimant une similarité totale et la valeur 0 une dissemblance totale. D'autre part la mesure de similarité employée rend compte de la structure relationnelle entre les entités, c'est-à-dire des prédicats.

E	pere	sexe	age	français	americain
E1	(paul,yves)	(yves,garcon)	(yves,13) (paul,33)	(paul)	
E2	(john,ann)	(ann,fille)	(ann,28)(john,50)		(john)

Une première similarité (que l'on appellera *simple*) entre deux entités est élaborée à partir d'un calcul simple et de constatations sur ce calcul. Reprenons l'exemple donné dans [Bis93] (voir tableau ci-dessus). Pour calculer la similarité simple entre *paul* et *john*, les occurrences de chacune des entités sont comptées en mémorisant la place de l'entité dans le prédicat :
Occ(paul) =((pere 1)(age 1)(français 1)).
Occ(john) =((pere 1)(age 1)(americain 1)).

La similarité simple (notée O-SIM) entre les deux entités est égale au rapport entre la longueur de l'intersection de leur liste d'occurrences et la longueur de la plus grande des deux listes: O-SIM(paul,john) = $\frac{2}{3}$ = 67%. Cette mesure vérifie les propriétés de l'indice de similarité défini en analyse de données (voir section 5.1).

À partir de ce calcul simple, un certain nombre d'observations ont permis de déterminer un calcul plus complexe. Ces observations sont les suivantes :

1. Chaque prédicat n'a pas la même valeur sémantique, il convient donc d'allouer un poids à ces prédicats et de prendre en compte ce poids dans le calcul de la similarité.

2. Il convient aussi, en plus de la place de l'entité dans le prédicat, de tenir compte de la ressemblance des valeurs. Par exemple, pour le prédicat *age*, (yves,13) et (paul,33) ne doivent pas être similaires mais la similarité doit dépendre de la proximité des valeurs 13 et 33.

3. Enfin, la similarité entre paul et john ne doit pas seulement dépendre des prédicats qui portent sur paul et john, mais doit aussi tenir compte des similarités entre yves et ann qui sont respectivement le fils et la fille de paul et john. Autrement dit, la similarité entre deux entités x et y doit prendre en compte les similarités de toutes les entités en relation avec x et y.

À partir de ces contraintes, une nouvelle mesure plus complexe a été définie. La similarité entre deux entités est résolue par un système d'équations linéaires. Il est d'abord défini une similarité partielle entre deux entités par rapport à chaque prédicat dans lesquels les entités apparaissent. Sont examinés tous les couples d'entités apparaissant au même endroit dans le même prédicat dans les deux exemples.

Rappelons que l'on ne compare que les entités à la même place pour un prédicat donné. La similarité partielle entre deux entités par rapport à chaque prédicat est définie de la manière suivante :
$SIM_{predicat}(Ar, Br) = (\frac{1}{n}\sum_{i=1}^{n} prox(Ai, Bi).\frac{1}{m}\sum_{i=1}^{m} prox(Vi, Ui)).Poids(P)$

Où r est le rang des deux entités Ar et Br dans le prédicat, n le nombre d'entités dans le prédicat. Si i = r alors prox(Ai,Bi) = 1 sinon prox(Ai,Bi) = R-Sim(Ai,Bi) où R-Sim(Ai,Bi) est la similarité relative entre deux entités. De la même manière m est le nombre de valeurs dans le prédicat, Vi et Ui sont les valeurs du prédicat dans l'exemple E1 et E2 respectivement et prox(Vi,Ui) = R-Sim(Vi,Ui). Pour simplifier le calcul, les poids des prédicats sont tous égaux à 1.

Pour notre exemple, en considérant tous les poids des prédicats égaux à 1, nous obtenons donc trois similarités partielles (pour le prédicat père, âge et sexe) :
$Sim_{pere}(paul, john) = \frac{1}{2}[prox(paul, john) + prox(yves, ann)]$
$Sim_{pere}(yves, ann) = \frac{1}{2}[prox(yves, ann) + prox(paul, john)]$
$Sim_{age}(yves, john) = [prox(yves, john)][SIM_{age}(13, 50)]$
$Sim_{age}(yves, ann) = [prox(yves, ann)][SIM_{age}(13, 28)]$
$Sim_{age}(paul, ann) = [prox(paul, ann)][SIM_{age}(33, 28)]$
$Sim_{age}(paul, john) = [prox(paul, john)][SIM_{age}(33, 50)]$
$Sim_{sexe}(yves, ann) = [prox(yves, ann)][SIM_sexe(garcon, fille)]$

Le système devient le suivant :
$Sim_{pere}(paul, john) = \frac{1}{2}[1 + SIM(yves, ann)]$
$Sim_{pere}(yves, ann) = \frac{1}{2}[1 + SIM(paul, john)]$
$Sim_{age}(yves, john) = 0,626262$
$Sim_{age}(yves, ann) = 0,84848$
$Sim_{age}(paul, ann) = 0,94949$
$Sim_{age}(paul, john) = 0,8282$
$Sim_{sexe}(yves, ann) = 0$

À partir de ce calcul sur les similarités partielles sur les prédicats entre les entités, il est introduit deux types de similarité. La première notée A-Sim est la similarité absolue entre deux entités, par rapport à un prédicat, elle correspond à la somme des similarités partielles par rapport à ce prédicat. Si np est le nombre de prédicats :
$A - Sim_{predicat}(X, Y) = \sum_{i=1}^{np} Sim_{predicat_i}(X, Y)$

La seconde similarité notée R-Sim est la similarité relative entre deux entités, elle correspond à la somme des similarités partielles par rapport à chaque prédicat, divisée par le maximum de la somme des poids pour chaque entité . Si npX est le nombre de prédicats dans lesquels est mise en jeu l'entité X, npY le nombre de prédicats dans lesquels l'entité Y est mise en jeu et Poids(Pi) le poids du prédicat i (toujours 1 dans notre exemple) alors :
R-Sim(X,Y) $= \frac{\sum_{i=1}^{np} Sim_{predicat_i}(X,Y)}{max(\sum_{i=1}^{npX} Poids(Pi), \sum_{i=1}^{npY} Poids(Pi))}$

On peut donc écrire la relation suivante entre R-Sim et A-Sim :
R-Sim(X,Y) $=\frac{A-Sim(X,Y)}{max(\sum_{i=1}^{npX} Poids(Pi),\sum_{i=1}^{npY} Poids(Pi))}$

D'où les similarités relatives entre les entités :
R-Sim(paul,john) $= \frac{Sim_{pere}(paul,john)+Sim_{age}(paul,john)}{3}$
R-Sim(yves,ann) $= \frac{Sim_{pere}(yves,ann)+Sim_{age}(yves,ann)+Sim_{sexe}(yves,ann)}{3}$
R-Sim(yves,john) $= \frac{Sim_{age}(yves,john)}{3}$
R-Sim(paul,ann) $= \frac{Sim_{age}(paul,ann)}{3}$

Dans l'exemple précédent, la similarité de valeurs pour les prédicats age et sexe sont définies de la manière suivante :
$Sim_{age}(V1,V2) = (99- \mid V1 - V2 \mid)/99$
et $Sim_{sexe}(V1,V2) = 1$ si sexe(V1) = sexe(V2) et 0 sinon.

Ce système est résolu par la méthode de Jacobi, c'est-à-dire en initialisant les valeurs de R-Sim puis en procédant à un calcul itératif jusqu'à une précision ε donnée. Dans l'exemple précédent, les similarités prennent les valeurs :
R-Sim(paul,john) = 0,515
R-Sim(yves,ann) = 0,5354
R-Sim(yves,john) = 0,208
R-Sim(paul,ann) = 0,316
et
A-Sim(paul,john) = 1,545
A-Sim(yves,ann) = 1,602
A-Sim(yves,john) = 0,624
A-Sim(paul,ann) = 0,948

Rappelons la définition des exemples E1 et E2 :
E1 = pere(paul,yves) & sexe(yves,garcon) & age(yves,13) & age(paul,33) & français(paul).
E2 = pere(john,ann) & sexe(ann,fille) & age(john,50) & age(ann,28) & americain(john).

Soit $Poids(E) = \sum_{ent}^{entits(E)} \sum Poids(Occ(ent))$, c'est-à-dire la somme pour toutes les entités de l'exemple E du poids total de chaque entité. Soit $Sim_p(E1,E2)$ la somme intermédiaire des similarités pour chaque entité mise en jeu dans le calcul. Soit Sim(E1,E2) la similarité totale entre l'exemple E1 et l'exemple E2.

Soit LE1 = liste des entités de E1 et LE2 = liste des entités de E2. Pour calculer la similarité entre deux exemples, l'algorithme suivant est défini :

1. $Sim_p(E1, E2) = 0$

2. **Tant que** (LE1 et LE2) $\neq \emptyset$ **faire**

   (a) Chercher les entités X $\in$ LE1 et Y $\in$ LE2 tels que R-Sim(X,Y) soit maximale

   (b) $Sim_p(E1, E2) = Sim_p(E1, E2)$+A-Sim(X,Y).R-Sim(X,Y)

   (c) Supprimer X et Y respectivement de LE1 et LE2

3. **fin tant que**

4. $Sim(E1, E2) = \frac{Sim_p(E1,E2)}{max(Poids(E1),Poids(E2))}$

On peut réécrire la formule $Sim_p(E1, E2) = Sim_p(E1, E2)$+A-Sim(X,Y).R-Sim(X,Y) en remplaçant R-Sim(X,Y) par $\frac{A-Sim(X,Y)}{max(\sum Poids(Occ(X)),\sum Poids(Occ(Y)))}$

Ce qui nous donne :

$$Sim_p(E1, E2) = Sim_p(E1, E2) + \frac{A-Sim(X,Y)^2}{max(\sum Poids(Occ(X)),\sum Poids(Occ(Y)))}.$$

Les poids de tous les prédicats étant égaux à 1, en appliquant l'algorithme précédent aux deux exemples E1 et E2 cela donne :
LE1 = {paul,yves}, LE2 = {john,ann}.
$Sim_p(E1, E2) = 0$.
X = yves et Y = ann (valeur de similarité maximale)
$Sim_p(E1, E2) = 0 + \frac{A-Sim(yves,ann)^2}{max(3,3)} = \frac{1{,}602^2}{3} = 0.855$
LE1 = {paul} et LE2 = {john}
X = paul et Y = john (dernier choix possible)
$Sim_p(E1, E2) = 0.855 + \frac{A-Sim(paul,john)^2}{max(3,3)} = 0.855 + \frac{1{,}545^2}{3} = 1.650$

LE1 = $\emptyset$ et LE2 = $\emptyset$.

Remarquons que dans ce cas LE1 et LE2 sont vides en même temps car E1 et E2 possèdent le même nombre d'entités. Ce n'est pas forcément toujours le cas, l'algorithme s'arrête dès que l'un des deux ensembles est vide. Calculons le poids de chaque exemple :

Poids(E1) = $\sum Poids(Occ(paul)) + \sum Poids(Occ(yves)) = 3+3 = 6$

Poids(E2) = $\sum Poids(Occ(john)) + \sum Poids(Occ(ann)) = 3+3 = 6$

La similarité totale entre les deux exemples est donc :
$SIM(E1, E2) = \frac{1.650}{max(6,6)} = 0.275$

Conclusion sur la mesure de KBG

Le calcul d'une similarité entre deux exemples est assez complexe. Cette complexité cependant est justifiée par le fait que la mesure utilisée rend compte des relations entre les entités ainsi que des poids des prédicats utilisés. Dès que beaucoup de connaissances sont prises en compte dans la similarité, celle-ci se complique rapidement. La mise en œuvre de cet algorithme s'avère pourtant rapide et la méthode de Jacobi (par itérations successives) converge assez rapidement (quelques itérations) et la complexité totale est polynomiale.

Dans le chapitre 6, nous comparerons en détail la mesure de KBG avec celle utilisée dans ANAÏS.

5.4 Mesures de similarité dans le raisonnement fondé sur la mémoire

5.4.1 MBRtalk

Le système MBRtalk [SW86] est un système expérimental de raisonnement fondé sur la mémoire. MBRtalk est un système qui renvoie la prononciation d'un mot anglais. La base de données utilise environ 4438 mots. L'approche abordée est différente de celle du RBC, car en raisonnement fondé sur la mémoire (Memory-Based Reasoning ou MBR) l'extraction se fait par rapport à tous les cas de la mémoire. Il n'y a pas véritablement de systèmes d'indexation et la mesure de similarité employée est fondée sur un calcul statistique. Pour pallier le manque d'efficacité éventuel dû à la comparaison avec tous les cas, une machine parallèle (la « connection machine system» est un calculateur parallèle SIMD avec 2^{16} processeurs). Dans MBRtalk, les étapes sont les suivantes :

1. Compter le nombre de cas possédant une même caractéristique (par exemple le nombre de cas possédant un même symptôme en diagnostic médical).
2. Utiliser le calcul précédent pour produire une mesure de similarité
3. Calculer les différences entre les cas de la mémoire et le cas courant
4. Trouver le cas le plus proche du cas courant.

L'application de MBRtalk se situe dans le domaine du diagnostic médical et dans ce domaine, la notion de probabilité d'apparition d'une maladie est souvent utilisée. La mesure de similarité produite dépend de cette probabilité. La probabilité que la maladie A apparaisse avec un symptôme donné B est donnée par : $P = \frac{NAB}{NB}$ où NB est le nombre de cas ayant pour symptôme B, NAB le nombre de cas ayant comme symptôme B et comme diagnostic de maladie A. À chaque caractéristique est associé un poids qui correspond à la probabilité qu'il s'agisse de la maladie du cas courant avec cette caractéristique. À l'aide de ce poids, la distance entre la valeur de la caractéristique du cas de la mémoire et du nouveau cas est calculée, puis une distance globale entre le nouveau cas et le cas courant est calculée.

Les n (paramétrable) cas ayant la plus petite distance avec le cas sont sélectionnés. Quatre résultats sont possibles :

- Aucun cas n'est suffisamment proche du cas courant. Le système sait qu'il ne peut émettre un diagnostic.
- Un très petit nombre de cas est sélectionné avec le même diagnostic. Le système donne un diagnostic probable.

– Un nombre significatif des cas est sélectionné, ces cas ayant le même diagnostic. Le système donne alors le diagnostic en précisant qu'il est très probable.

– Il y a plusieurs diagnostics possibles à partir des cas sélectionnés. Le système peut alors acquérir de nouvelles connaissances. Il propose le diagnostic le plus probable ou propose de travailler avec un diagnostic incertain.

5.4.2 Similarité dans le raisonnement basé sur les cas

Choix des meilleurs cas

Les méthodes d'indexage, les organisations de la mémoire et les algorithmes d'extraction peuvent sélectionner des cas appariés partiellement, mais en sélectionnant un très grand nombre de cas, il faut donc choisir les meilleurs. Un comptage par poids sur les attributs dépend du contexte. Plusieurs méthodes sont utilisées : des heuristiques de préférence, une analyse dimensionnelle et des fonctions d'évaluation pondérées éventuellement modifiables.

Choisir le meilleur cas nécessite d'être capable d'apparier deux cas pour générer une correspondance entre leurs parties. L'appariement est simple dans le cadre d'une représentation par attributs, cependant dans certains domaines comme la planification, il est inévitable de considérer des représentations structurelles ou relationnelles et cela complique fortement le processus d'appariement. Il est nécessaire d'avoir plus d'informations que les caractéristiques des cas, par exemple les relations entre les parties du cas et les relations avec les autres cas. La similarité est alors dite structurelle et devient plus complexe à évaluer.

Dans la mesure où dans ANAÏS, les tâches communes sont déjà disponibles, il ne reste qu'à évaluer l'importance relative de chacune de ces tâches et d'évaluer ainsi une similarité globale. Le problème de la similarité, qui apparaissait au départ comme complexe, est alors ramené, après application de l'algorithme précédent, à une mesure de similarité de surface sur les tâches communes trouvées. Pour cette raison, les mesures de similarité examinées sont celles qui sont le plus couramment utilisées en RBC et sont généralement des mesures de surface.

La plupart des systèmes RBC évaluent la similarité de deux cas en comptabilisant les caractéristiques communes. Les poids associés à ces caractéristiques dépendent de leur importance relative donnée par l'expert. Dans les systèmes CYRUS [RS89a], SHRINK [Kol85] et JULIA [Kol88] les similarités sont liées au système d'indexation. Dans SHRINK et JULIA, un second système d'indexation permet de modifier l'importance relative des caractéristiques en fonction des erreurs signalées par l'expert. Dans MEDIATOR [Sim85], un premier filtrage permet d'éliminer les cas ne partageant pas les caractéristiques majeures et les autres cas sont ré-évalués vis-à-vis d'un sous-ensemble des caractéristiques les plus importantes. Le processus d'adaptation de la solution extraite peut éventuellement modifier l'évaluation de la similarité. HYPO [AR88] sélectionne les cas les plus proches, la similarité s'évaluant par les faits spécifiques et par le rôle que joue le cas dans le raisonnement. GREBE [Bra89] utilise une explication pour évaluer la similarité des cas.

Nous avons observé différents points autour desquels les systèmes RBC se différencient :

- L'extraction de cas et l'évaluation de la similarité peuvent être confondues ou différenciées. Dans le cas où ces étapes sont bien séparées, les cas sont extraits en fonction de leur similarité et le processus d'extraction est indépendant. Une évaluation supplémentaire est donc nécessaire pour déterminer le cas le plus similaire.

- Les cas sont représentés à l'aide de caractéristiques simples ou abstraites. Lorsque les caractéristiques sont abstraites, leur extraction est un processus externe au système.

- Soit les caractéristiques s'expriment de la même manière dans tous les cas de la mémoire, soit elles ne sont pas exprimées de la même manière dans les différents cas de la mémoire, un processus d'équivalence de caractéristiques est alors nécessaire.

- Dans certains systèmes [Don89], pour évaluer la similarité, la connaissance du domaine est nécessaire en plus des caractéristiques.

- Des fonctions de similarité numériques ou heuristiques sont utilisées pour déterminer un ordre partiel sur les cas extraits.

Un tel éventail de possibilité nous invite à observer plus en détail certains systèmes ayant des méthodes de calcul de similarité plus ou moins complexes.

Similarité dans JUDGE

Le programme JUDGE [Bai86] travaille dans le domaine des sentences criminelles. Il modélise un jugement qui détermine les sentences pour les personnes soupçonnées de crimes. L'entrée est une description du cas, qui inclut la charge, les événements qui ont eu lieu, et les statuts légaux vis-à-vis des crimes de cette nature, e.g les différents niveaux d'emprisonnements permis et les conditions associées. JUDGE commence avec une bibliothèque de cas vide et des heuristiques pour déterminer les sentences quand aucun cas n'est applicable à la nouvelle situation. Il construit ensuite une base de cas à partir de ses propres décisions. Un nouveau crime est d'abord interprété en évaluant les événements sérieux, l'intentionnalité, la justification, etc. Ensuite le cas interprété est utilisé pour retrouver des cas similaires dans la bibliothèque. Finalement la sentence stockée dans le cas extrait est adaptée au crime courant en renforçant ou en allégeant la sentence selon la comparaison effectuée avec l'ancien crime.

La mesure de similarité est fondée sur une interprétation des événements selon les critères de gravité, de volonté et de justification. Elle est utilisée dans la phase d'extraction et phase d'adaptation (la sentence est calculée à l'aide de la sentence extraite et à l'aide des différences). Chaque caractéristique du cas est comparée avec les caractéristiques des cas de la mémoire. Soit la caractéristique est identique, soit elle est mauvaise. Tant que le nombre de caractéristiques non partagées est en dessous d'un certain seuil, on continue à apparier les autres caractéristiques. Dès que le seuil est atteint, on passe au cas suivant dans la mémoire. Tous les cas possédant un nombre minimal de caractéristiques communes sont étudiés et on applique des stratégies de choix pour sélectionner le meilleur. D'abord le système préfère les nouveaux cas, puis les cas récents, puis le système assigne différents poids aux caractéristiques et réfère les cas ayant le meilleur score. Les caractéristiques sont par exemple : utilisation d'un cran d'arrêt, agression avant le meurtre, représailles de l'agressé, violence progressivement plus forte, préméditation.

La mesure utilisée est donc un modèle lexicographique en psychologie cognitive. Cette mesure est cependant complétée par un ensemble d'heuristiques pour choisir le meilleur cas, là aussi, il y a convergence avec les travaux en psychologie cognitive. Le modèle psychologique convergent dépend évidemment du domaine d'application, ici celui de la justice.

Similarité dans OGRE

OGRE [Don89] est un système RBC indépendant du domaine d'application. Ce système, déjà mentionné dans la partie 3.2, a été testé dans le domaine de la fabrication de semi-conducteurs. Deux types de connaissances sont distinguées : les cas et les facteurs. Chaque cas est caractérisé par les facteurs, qui sont décrivent les caractéristiques du domaine. Un domaine est caractérisé avec une collection de facteurs-descripteurs et la similarité entre deux expériences est déterminée par la méthode d'appariement spécifiée pour chaque facteur et par le poids associé à ce facteur. Le modèle psychologique sous-jacent est le modèle compensatoire dans lequel pour chacune des caractéristiques, une méthode peut être spécifiée (pour plus de détails voir section 3.2).

Similarité dans CASEY

CASEY [Kot88] utilise le RBC pour se rappeler les problèmes passés et utilise un modèle causal pour justifier la réutilisation des solutions précédentes et pour résoudre de nouveaux problèmes. Le domaine d'application est celui du diagnostic médical et en particulier des problèmes cardiaques. La comparaison de deux cas se fait suivant ses caractéristiques. Cependant, une des originalités de CASEY est de considérer que l'importance de chaque caractéristique est différente d'un cas à l'autre. Par exemple, le rythme cardiaque peut être important et le pouls pas très important dans un cas, l'inverse étant possible dans un autre cas.

La mesure de similarité permet donc de déterminer les caractéristiques importantes pour chaque cas.

Pour déterminer les caractéristiques importantes du problème à résoudre, CASEY se réfère au modèle causal du domaine. Les caractéristiques importantes du cas extrait et du nouveau problème sont alors comparées. Le système évalue ensuite les différences entre le cas extrait et le nouveau problème et adapte ensuite la solution en utilisant de nouveau le modèle causal du domaine. Pour déterminer l'importance d'une caractéristique dans un cas, CASEY utilise deux informations : une priorité et un compteur, qui est incrémenté à chaque fois que la caractéristique est utilisée dans le modèle causal. Le quotient priorité/compteur détermine l'importance de la caractéristique dans le cas. La similarité entre deux cas est ensuite déterminée par le nombre de caractéristiques communes et leurs poids associés.

Dans ce système, le modèle sous-jacent au calcul de la similarité est donc identique à ceux vus précédemment. La différence essentielle est dans la détermination de l'importance d'une caractéristique, qui est en général fixe pour tous les cas, mais qui est redéterminée ici en fonction du modèle causal du domaine. Cela exige d'avoir déjà une grande connaissance du domaine. Or, la justification de l'utilisation d'un système RBC est en général le manque de connaissances et surtout de modèle de connaissances du domaine. Dans CASEY, le RBC est donc utilisé en supplément du mode de raisonnement, appelé « raisonnement fondé sur un modèle » (ou « model-based reasoning»).

Dans le chapitre suivant, nous allons comparer les différentes mesures étudiées afin de discuter ce qu'il y a lieu de retenir dans ANAÏS vis-à-vis de nos objectifs.

Chapitre 6

Mesure de similarité dans ANAÏS

6.1 Comparaison des différentes mesures et évaluation par rapport aux contraintes dans ANAÏS

6.1.1 Comparaison des mesures de similarité

Il y a en quelque sorte une dualité dans les mesures utilisées dans OGRE et dans CASEY. Dans le premier système, ce sont les méthodes d'appariement entre caractéristiques qui font partie des connaissances initiales, l'importance de chaque caractéristique étant fixe. Dans le second à l'inverse, il n'y a pas de méthodes d'appariement dépendantes de la caractéristique (puisque la similarité finale dépend du nombre de caractéristiques communes) mais les connaissances initiales sont utiles pour le calcul de l'importance. Dans OGRE, on mesure la similarité entre valeurs alors que dans CASEY, on pondère dynamiquement les caractéristiques.

Tout dépend donc de la quantité de connaissances disponibles *a priori* dans le domaine. Plus le modèle de connaissances est riche, plus la similarité entre deux cas sera précise. De même, plus le modèle des connaissances est pauvre, plus la mesure de similarité sera imprécise. Notons toutefois que dans la mesure de MBRtalk, la connaissance sur le domaine est limitée puisqu'il s'agit uniquement de décrire les symptômes et les maladies pour un cas donné. Rien ne permet d'affirmer cependant, que cette mesure est plus éloignée de la réalité. Il y a un glissement du critère « proche de la réalité » vers le critère « efficacité ».

À partir de cette étude sur les différentes mesures de similarité, plusieurs voies de recherche peuvent être engagées. Il est important, par exemple, d'étudier l'influence globale de la similarité dans les processus RBC et de savoir comment évaluer la similarité entre les cas lorsque les caractéristiques ne sont pas décrites dans le même formalisme. Le rôle joué par la connaissance du domaine dans l'évaluation de la similarité est également un problème très important. Dans KBG [Bis93], par exemple, la prise en compte de la structure des exemples (relations entre les entités) permet d'exploiter au maximum les connaissances du domaine.

6.1.2 Évaluation par rapport aux objectifs d'ANAÏS

L'objectif inhérent à ANAÏS est de déterminer une mesure entre « atomes », ces atomes étant des tâches structurées dans un modèle arborescent. Un certain nombre de remarques vont nous aider à déterminer la mesure de similarité dans ANAÏS :

1. Comme dans JUDGE, l'importance de chaque atome devra être déterminée, un simple comptage des tâches en commun serait tout à fait superficiel. Une tâche peut en effet être très complexe ou relever d'un simple calcul algébrique, ce qui lui confère une importance tout à fait différente. Par contre, il est tout à fait inopportun de reconsidérer cette importance par rapport à chaque cas examiné, comme il est fait dans CASEY, une tâche est en effet plus ou moins importante par rapport à la base de connaissances complète. Le poids d'une tâche est a priori fixe dans une base fixée, mais lorsque la base sera amenée à être mise à jour, le poids de cette tâche sera amené lui aussi à être re-calculé.

2. La mesure de similarité d'ANAÏS ne doit intervenir que lors de la phase de sélection et sur un certain nombre de cas extraits, donc

peu nombreux. Dans la plupart des systèmes CBR, la mesure de similarité intervient aussi dans la construction et modification du système d'indexation et doit donc répondre à deux critères importants : l'efficacité (se rappeler vite) et la structuration des connaissances (se rappeler les connaissances pertinentes). Réaliser un bon compromis entre ces deux critères relève parfois d'une étude complexe en fonction de la représentation des connaissances utilisée. Dans ANAÏS, le critère d'efficacité perd de son poids, car peu de cas sont examinés et le calcul n'est pour l'instant utilisé que dans la phase de sélection.

3. Dans ANAÏS, tenir compte des informations relationnelles est indispensable pour traiter la ressemblance des cas. La « structure » des cas est ici une décomposition de tâche en sous-tâches, elle est du type problème/sous-problème. Ayant déterminé l'ensemble des tâches communes au cas et au problème à résoudre, il est donc nécessaire de déterminer l'importance des tâches et ce, indépendamment du domaine utilisé. En effet, l'environnement de résolution de problèmes utilisé n'est pas fonction du domaine d'application, mais est destiné justement à recevoir des problèmes de toute application. En fonction de la place d'une tâche dans la base, donc de sa décomposition, un modèle d'affectation automatique d'importance est proposé (voir 6.2).

4. Dans le calcul final de la similarité, il est important de tenir compte, en plus des tâches pondérées communes, des tâches totales pondérées des deux entités comparées (c'est-à-dire dans ANAÏS du raisonnement partiel RPi et du cas complet). Nous reprendrons donc le même système que dans KBG qui consiste à diviser le poids des prédicats en commun par le poids total de tous les prédicats (voir section 6.3).

En fonction de ces remarques de fond, il apparaît que la mesure de similarité dans KBG soit adéquate pour notre système. Dans KBG, on commence par compter le nombre d'occurrences communes et la mesure de similarité entre deux entités dépend du nombre de prédicats (pondérés) communs par rapport au nombre total de prédicats (pondérés également).

Dans ANAÏS, nous commençons par rechercher toutes les tâches communes, ce qui est un parallèle avec la recherche de tous les prédicats communs. À partir de ces tâches communes, il nous faut définir une mesure de similarité dépendant de l'importance de chaque tâche. Parallèlement au poids des prédicats dans les exemples de KBG, nous définissons (voir section 6.2) un algorithme permettant de déterminer l'importance (autrement dit le poids) de chaque tâche dans la base de tâches.

Dans KBG, lorsque l'on calcule la similarité entre deux prédicats pour deux entités données, on tient compte de la ressemblance des valeurs dans les prédicats (exemple : dans age(yves,13) et age(paul,33) la similarité entre yves et paul dépend de la comparaison des valeurs 13 et 33). Dans ANAÏS, les valeurs à comparer sont les entrées et les sorties des tâches. La comparaison entre valeurs est discutée dans la section 6.3 mais pour l'instant non effectivement prise en compte.

De plus, dans KBG, la similarité entre deux entités dépend des similarités avec les autres entités qui sont en relation avec elles. Dans notre modèle de tâches, chaque tâche est en relation avec, soit la tâche hiérarchiquement supérieure, soit avec ses sous-tâches. La prise en compte de l'aspect relationnel est donc implicite et ne nécessite pas de calcul particulier. Ce dernier est en effet implicitement lié à l'algorithme qui détermine l'importance de chaque tâche (section 6.2).

Bien que les objectifs de KBG (classification conceptuelle de données) et que le langage utilisé (sous-ensemble de la logique du premier ordre) soient éloignés de notre contexte, la mesure employée est tout à fait utilisable dans ANAÏS via une adaptation au contexte de notre modèle de tâches.

6.2 Évaluation des importances

C'est la première étape nécessaire pour donner une valeur de similarité entre deux cas. En fonction des observations faites sur les bases de connaissances réelles implémentées dans SCAI, un certain nombre de règles ont été dégagées :

- Règle 1 : plus la tâche a de générations filles (profondeur), plus elle est complexe et plus son importance est grande.

– Règle 2 : plus la tâche a de filles directes (largeur), plus elle est complexe et plus son importance est grande.

– Règle 3 : Les tâches récursives sont plus importantes que les autres tâches ayant le même nombre de générations. Le nombre d'appels récursifs de la même tâche correspondra au nombre de générations propres de la tâche.

Outre ces règles observées, une contrainte sur les valeur des poids des tâches par rapport à leur décomposition est triviale : la somme des poids des sous-tâches d'une tâche principale doit être égale au poids de la tâche principale. À partir de cette constatation, un algorithme d'affectation a été mis au point réalisant les contraintes énoncées ci-dessus. Cet algorithme part des feuilles (tâches terminales) et remonte dans l'arborescence des tâches en affectant à chaque fois la fonction F de la manière suivante :

1. Pour toutes les feuilles (tâches terminales) de l'arborescence, affecter p à F où p est une valeur paramétrable fixée par l'utilisateur.

2. Si le nœud est une tâche séquentielle non récursive alors F(n) = $k \times \sum_{i=1}^{nombredefils} F(i)$ où k représente le facteur relatif entre la profondeur et la largeur fixé par l'utilisateur.

3. Si le nœud est une tâche de choix non récursive alors F(n) = $k \times Max_{i=1}^{nombre\ de\ sous-taches}(F(i))$

Le facteur k, fixé par l'utilisateur, permet de moduler l'importance relative de la profondeur et de la largeur. Le poids de la tâche (c'est-à-dire F(n)) est multiplié par k à chaque remontée dans l'arbre. Donc F(n) est directement proportionnel à $k^{profondeur\ du\ noeud}$. Lorsque k est inférieur à 1, plus le nœud est profond (récursivement) et plus son importance est diminuée. Dans ce cas, la largeur prend le pas sur la profondeur. Lorsque k est supérieur à 1, le phénomène inverse se produit et le poids augmente en fonction de la profondeur totale. Dans notre exemple nous prendrons k=2, ce qui privilégie la profondeur par rapport à la largeur, et p = 0.5.

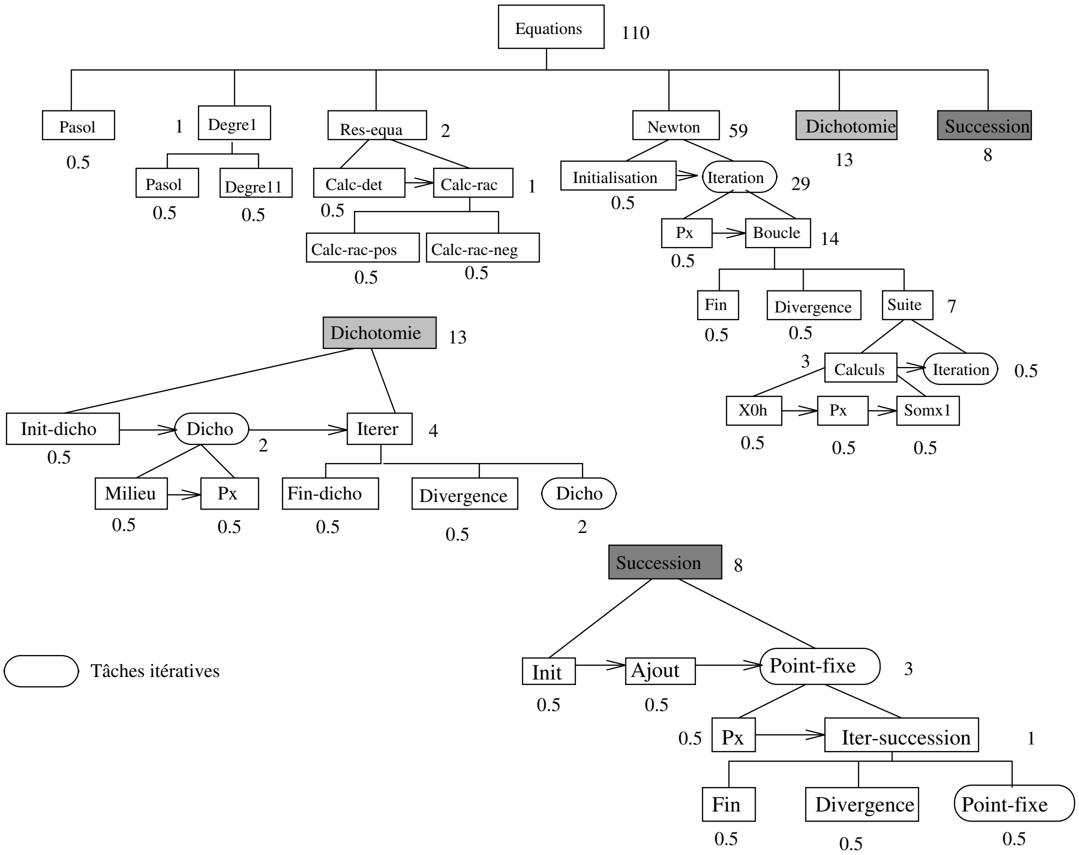


FIG. 6.1 - : Calcul de la largeur et la profondeur des nœuds. Les paramètres prennent les valeurs suivantes : p=0,5 et k =2

Pour illustrer cet algorithme, reprenons l'exemple précédent (voir figure 6.1). L'application des formules précédentes n'est valable que lorsque la tâche n'est pas récursive. Dans la figure précédente, il n'y a pas d'ambiguïté dans F(Dicho). En effet, même si celle-ci est appelée dans une autre tâche, la récursivité n'est pas directe.
F(Dicho)= $2 \times [\text{F(Milieu)} + \text{F(Px)}] = 2$.
Il suffit donc de reporter cette valeur dans la sous-tâche de la tâche *iterer*.

Lorsque la récursivité est directe, il n'est pas possible non plus d'appliquer les formules. Pour la tâche *point-fixe*, l'équation serait la suivante :
F(point-fixe) $= 2 \times [0{,}5 + 2 \times [\text{Max } (0{,}5\,;\ \text{F(point-fixe)})]]$.
En considérant 0,5 comme le maximum, on obtient F(point-fixe) $= 3$, ce qui est impossible puisque 0,5 doit être plus grand. En considérant F(point-fixe) comme maximum (ce qui est plus probable étant donné que *point-fixe* est une tâche récursive et donc doit être plus importante qu'une tâche terminale), on obtient F(point-fixe) $= -\frac{1}{3}$, ce qui est impossible aussi car notre hypothèse
Max(0,5 ; F(point-fixe)) = F(point-fixe) devient fausse.

La solution est de prendre deux valeurs différentes pour F(point-fixe). Lors du dernier appel de la tâche *point-fixe*, la tâche *iterer-succession* appellera soit la tâche *Fin* soit la tâche *Divergence*. Nous prendrons donc F(point-fixe) $= 0.5$ (comme les tâches terminales) et en reportant dans l'équation
F(point-fixe) $= 2 \times [0{,}5 + 2 \times [\text{Max } (0{,}5\,;\ \text{F(point-fixe)})]]$,
on obtient F(point-fixe) $= 3$.

6.3 Calcul final de la similarité

Les poids de chaque tâche étant déterminés, les tâches communes entre le cas et les raisonnements partiels du problème à résoudre (les RPi) étant calculées, la valeur de la similarité entre le cas et un RPi est la somme des similarités trouvées entre le cas et un RPi.

Dans l'évaluation de la similarité entre un raisonnement partiel (RPi) et un cas, il faut prendre en compte le fait que plus un cas est complexe et plus le RPi est simple, plus la valeur de similarité sera petite. Inversement, plus un RPi est complexe et plus un cas est simple, plus la valeur de similarité est grande. Plus simplement, si la complexité est la somme des importances des tâches, la similarité totale dépend du rapport des complexités entre le RPi et le cas.

La détermination de la valeur de similarité entre un cas et un RPi dépend donc : du poids global des tâches de RPi, du poids global des tâches du cas et du poids des tâches communes. Si P_{RPi} est le poids total des tâches du RPi, P_{cas} est le poids total des tâches dans le cas, Ntc le nombre total des tâches communes, les tâches en commun étant les tâches i alors la valeur de similarité structurelle entre le RPi et le cas est donnée par la formule :
$Sim_{struct}(cas, RPi) = \frac{Min(P_{RPi}, P_{cas})}{Max(P_{RPi}, P_{cas})} \times \sum_{i=1}^{Ntc} F(tachei)$.

Ce qui donne pour les exemples précédents :

```
(cas1 (newton   (initialisation
  ((iteration
     (px boucle
(suite
   (calculs
      (xOh px somx1))))) 17)
    iteration (px boucle (divergence)))))

(cas2 (dichotomie (init-dicho
                  ((dicho (milieu px) iterer) 19)
                  iterer (fin-dicho))))

(cas3 (succession
         (init ajout
             ((point-fixe (px iterer-succession)) 13)
       point-fixe (fin))))

(RP1 (xOh px somx1))
(RP2 (dicho (milieu px)))
(RP3 (px))
```

Tâches communes entre
cas1 et RP1 : (x0h px somx1).
cas1 et RP2 : ().
cas1 et RP3 : (px).
cas2 et RP1 : ().
cas2 et RP2 : (dicho (milieu px)).
cas2 et RP3 : (px).
cas3 et RP1 : ().
cas3 et RP2 : ().
cas3 et RP3 : (px).

Notons poids(cas) la somme des poids des tâches du cas et poids(RPi) la somme des poids des tâches de RPi. Soit $poids(cas) = \sum_{i=1}^{n} F(tache_i)$, n étant le nombre de tâche du cas et $tche_i$ chaque tâche du cas. De même, $poids(RPi) = \sum_{i=1}^{m} F(tache_i)$ si m est le nombre de tâches du RPi.

Pour l'exemple précédent on a les résultats suivants :
poids(cas1)=59+0.5+29+0.5+14+7+3+0.5+0.5+0.5+29+0.5+14+0.5=158.5
poids(cas2)=13+0.5+2+0.5+0.5+4+4+0.5=25
poids(cas3)=8+0.5+0.5+3+0.5+1+3+0.5=17
poids(RP1)=0.5+0.5+0.5=1.5
poids(RP2)=2+0.5+0.5=3
poids(RP3)=0.5

Notons que lorsqu'une tâche est sous-tâche de plusieurs tâches différentes, le poids choisi est le plus petit. Ce choix se justifie par le fait qu'on ne sait pas *a priori* dans quelle partie de la base de tâches se trouve la sous-tâche, nous prenons la similarité la plus petite afin d'obtenir plutôt une similarité plus petite.

Un autre problème ouvert subsiste toutefois dans le calcul du poids total d'un cas ou d'un RPi. Concernant les tâches itératives, nous avons conservé l'information du nombre d'appel de la tâche. Ce nombre peut être important, comme nous le verrons dans le domaine électromyographique (voir chapitre 7), ou totalement dénué d'intérêt comme dans le cas où l'on calcule itérativement la factorielle d'un entier. Dans notre exemple, ce nombre n'est pas pris en compte mais il est simple de modifier le calcul précédent pour l'intégrer. Soit n1 le nombre d'appels d'une tâche récursive dans un cas et n2 le nombre d'appels de la même tâche récursive dans un RPi. Lors du calcul des poids des tâches communes, il suffit lorsque l'on rencontre la tâche itérative de multiplier le poids de cette tâche par le rapport $Min(n1, n2) \div Max(n1, n2)$.

Une méta-connaissance sur les tâches du domaine s'avère donc indispensable dans le cas général. Dans le domaine électromyographique cette connaissance est donnée par le fait que la règle de simplification des tâches récursives n'est pas appliquée. En effet tous les appels successifs d'une tâche sont conservés dans la syntaxe des cas (le nombre d'appels d'une tâche récursive est rarement supérieur à 3).

Pour simplifier le calcul, la similarité entre chaque cas et chaque RPi ne dépend pas du nombre d'appels des tâches récursives.

Ces similarités sont les suivantes :

$Sim_{struct}(cas1, RP1) = (0.5 + 0.5 + 0.5) \times 1.5 \div 158.5 = 0.01419$
$Sim_{struct}(cas1, RP2) = 0$
$Sim_{struct}(cas1, RP3) = 0.5 \times 0.5 \div 158.5 = 0.00157$
$Sim_{struct}(cas2, RP1) = 0$
$Sim_{struct}(cas2, RP2) = (2 + 0.5 + 0.5) \times 3 \div 25 = 0.36000$
$Sim_{struct}(cas2, RP3) = 0.5 \times 0.5 \div 25 = 0.01000$
$Sim_{struct}(cas3, RP1) = 0$
$Sim_{struct}(cas3, RP2) = 0$
$Sim_{struct}(cas3, RP3) = 0.5 \times 0.5 \div 17 = 0.01470$

Si r le nombre de RPi, notons RP = $\sum_{i=1}^{r} RPi$ et $Sim_{struct}(cas, RP) = \sum_{i=1}^{r} Sim_{struct}(cas, RPi)$ Les résultats finaux sont les suivants :
$Sim_{struct}(cas1, RP) = 0.01576$
$Sim_{struct}(cas2, RP) = 0.37000$
$Sim_{struct}(cas3, RP) = 0.01470$.

Ces valeurs ne correspondent pas à un pourcentage de similarité, car si une tâche commune est trouvée, son poids est entièrement pris en compte même si des sous-tâches de celles-ci sont aussi partagées. Il est indispensable d'ajouter le poids de ces sous-tâches car lorsqu'une tâche est résolue, elle peut l'être de différentes manières. S'il se trouve qu'elle est résolue de la même manière dans le cas et le problème à résoudre, la ressemblance en est forcément augmentée. Bien sûr, la valeur de similarité totale obtenue n'a plus rien d'un pourcentage de ressemblance et sa valeur en soi n'est pas très informationnelle. C'est dans la comparaison des différentes valeurs de similarité obtenues que l'information apparaît. Le but de ces calculs est en effet de classer les cas extraits par ordre de similarité. Dans cet exemple la résolution partielle du problème à résoudre est plus proche du cas2.

Jusqu'à maintenant le calcul de la similarité est purement structurel mais il convient dans l'évaluation de « ressemblance » entre deux problèmes résolus de tenir compte de la similarité entre les valeurs des attributs d'entrées/sorties des tâches. Bien sûr dans cette seconde évaluation, seules les tâches communes doivent être examinées.

La similarité finale entre un schéma de raisonnement partiel RP (ensemble des RPi) et un schéma de raisonnement complet (un cas) est alors définie :
$Sim(RP, cas) = \alpha * Sim_{struc}(RPis, cas) + \beta * Sim_{ES}(RPis, cas)$

Où $Sim_{struc}(RPis, cas)$ est la similarité structurelle calculée précédemment, $Sim_{ES}(RPis, cas)$ la similarité globale entre les valeurs d'entrées/sorties des tâches communes entre les RPi et le cas, et α, β deux coefficients multiplicateurs variables suivant le contexte. Si l'utilisateur ne désire qu'une similarité structurelle, il suffit de prendre $\beta = 0$.

6.3.1 Discussion sur la détermination de Sim_{ES}

La détermination de $Sim_{ES}(RPis, cas)$ n'est pas prise en compte dans cet exemple. Il suffit cependant de calculer la similarité entre chaque valeur d'attribut et de faire une somme éventuellement pondérée de ces similarités. La pondération pourra être déterminée par l'utilisateur en fonction de la plus grande importance de tel ou tel attribut, et/ou calculée par défaut en fonction de la complexité du typage des attributs.

Il est simple de comparer deux types comme des entiers, réels, caractères et chaînes de caractères. Pour calculer la similarité entre deux types simples comme les entiers ou les réels, il faut connaître la valeur maximale possible $Vmax$ pour l'attribut en question. La similarité peut alors définie comme suit : $1 - (r1 - r2)/Vmax$. Pour deux chaînes de caractères, elles sont soit identiques (similarité=1) soit différentes (similarité=0). Toutefois chacun des types complexes doit être examiné de manière indépendante.

Ainsi, il n'est pas aisé de calculer la similarité entre deux matrices. Il faut définir des opérations mathématiques complexes sur les matrices. La différence entre deux matrices apparaît comme une opération naturelle pour ce genre de calcul. De cette manière, deux matrices sont d'autant plus proches que leur différence est proche de la matrice nulle. Il est possible ensuite de déterminer une valeur de similarité proportionnelle à la taille de la matrice et inversement proportionnelle à la somme des valeurs de leurs différences :
$Sim_{ES}(M1, M2) = n * m \div \sum_{i=1}^{n} \sum_{j=n}^{m} (M1_{ij} - M2_{ij})$ Où n est le nombre de lignes et m le nombre de colonnes de la matrice.

De même lorsqu'un attribut est une liste de valeurs dans un domaine bien déterminé (par exemple l'attribut *lesion* de la classe conclusion dans le domaine électromyographique a pour domaine de valeurs : axonale, demyelenisant, mixed et indeterminee), la similarité entre les deux attributs est déterminée de la manière suivante :
nombre de valeurs communes $\div$ *nombre de valeurs total possible.*

Dans le cas où les instances sont des objets complexes (un attribut de l'instance est instance d'une autre classe), le calcul est fait de manière récursive. La valeur de similarité entre deux attributs revient à calculer la similarité entre deux instances. Ce calcul récursif permet de se ramener finalement à un calcul de similarité entre deux types simples.

6.3.2 Sélection du cas le plus similaire

Une fois le cas le plus analogue, donc ayant la plus grande valeur de similarité trouvée, la résolution complète de celui-ci est proposée à l'utilisateur. La phase d'adaptation qui existe dans certains systèmes RBC n'existe pas dans ANAÏS pour la bonne raison qu'il n'y a pas moyen d'évaluer les différences entre le cas analogue et le problème à résoudre, car ce dernier n'est qu'une succession de tâches résolues correspondant à une solution partielle.

Fournir le cas analogue peut cependant aider à résoudre complètement le problème. Le cas analogue extrait peut en effet avoir été résolu par un autre utilisateur. Par ailleurs, les tâches spécifiées dans la solution partielle du problème à résoudre peuvent être issues de plusieurs bases de tâches différentes. Plusieurs cas analogues peuvent alors être fournis et la solution du problème à résoudre peut alors s'inspirer des deux solutions, ce qui est tout à fait innovateur et permet de favoriser la créativité.

6.4 Modification incrémentale de la base de tâches

En suivant notre hypothèse de départ, l'incomplétude de la base de connaissances, et en constatant qu'il est souvent plus aisé d'exprimer la résolution de nouveaux cas que de modifier la base de connaissances, nous allons étudier le problème de la modification incrémentale de la base de tâches.

Dans un premier temps, les opérateurs élémentaires de modification sont décrits. A partir de nouveaux problèmes simples à insérer dans la base de connaissances, sont ensuite dégagées les séquences d'opérateurs à appliquer. Puis une réflexion sur l'application de ces opérateurs à des cas plus complexes est exposée.

6.4.1 Opérateurs élémentaires

Les opérations de base de modification de la base de tâches sont les suivantes(voir figure 6.2) :

1. Sur une tâche séquentielle : soit ajouter un fils, soit transformer la tâche séquentielle en une tâche de choix.

2. Sur une tâche de choix (automatique ou inter-active) : soit ajouter un choix supplémentaire, soit ajouter une séquence avant ou après les différents choix.

3. Sur une tâche terminale : soit ajouter un frère (cette opération est équivalente à, soit ajouter un fils à une tâche séquentielle si la tâche mère est séquentielle, soit ajouter un choix ou une séquence si la tâche mère est une tâche de choix.), soit transformer

la tâche terminale en une tâche séquentielle, ou encore, transformer la tâche terminale en une tâche de choix.

6.4.2 Séquence d'opérateurs à appliquer sur des cas simples

À partir d'une base de tâches donnée et d'un nouveau problème résolu, il reste à trouver l'ensemble des opérateurs à appliquer afin que la base de tâches intègre le nouveau problème. La recherche de la suite des opérateurs n'est pas un problème simple dans le cas général. Cependant, nous avons identifié un certain nombre de cas où cette recherche est triviale.

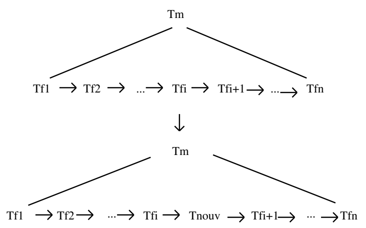
Les cas simples sont les cas où un ensemble de tâches nouvelles, les unes à la suite des autres, sont insérées dans la base de tâches. Il suffit alors de regarder la tâche mère et d'y ajouter les tâches correspondantes. Lorsque la tâche mère est une tâche séquentielle, on se ramène au premier type d'opérateur de la figure 6.2, mais au lieu d'ajouter un seul choix qui est une tâche terminale, on ajoute un choix qui est une tâche séquentielle (voir figure 6.3).

Lorsque toute une séquence de tâches nouvelles est ajoutée et que la tâche mère est une tâche de choix, il suffit d'ajouter un nouveau choix, ce nouveau choix étant une tâche séquentielle. Si la séquence de tâches nouvelles a pour tâche mère une tâche terminale, il suffira de transformer la tâche terminale en une tâche séquentielle. Les cas simples sont donc ceux où seules des nouvelles tâches sont introduites, mais où l'ordre d'exécution des tâches existantes n'est pas perturbé. Dans un travail ultérieur, il faut examiner les cas où il n'y a pas de nouvelles tâches mais où l'ordre d'exécution du nouveau problème ne suit pas les décompositions de la base de tâches.

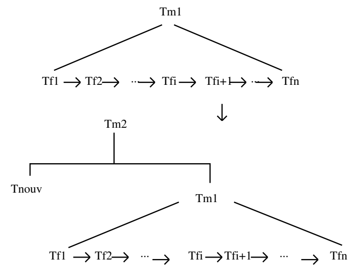
Ainsi, lorsque l'ordre d'appel des tâches n'est pas remis en question, il suffit de repérer toutes les nouvelles tâches dans le problème. Chaque nouvelle tâche est alors créée en y spécifiant les entrées/sorties, les sous-tâches éventuelles et le passage des entrées/sorties entre la nouvelle tâche et chacune des sous-tâches. Le traitement est récursif, c'est-à-dire que chacune des sous-tâches est elle-même traitée si elle est nouvelle. L'algorithme est le suivant :

```
procedure creer (T : tâche)
Début
    Si T n'existe pas alors
        Spécifier les entrées/sorties de T
        Spécifier les sous-tâches ST de T
        Pour chaque ST faire
            Spécifier le passage des entrées de T vers ST
            Spécifier le passage des sorties de ST dans T
            creer(ST)
        FinPour
    FinSi
Fin
```

Opérateurs élémentaires sur une tâche séquentielle

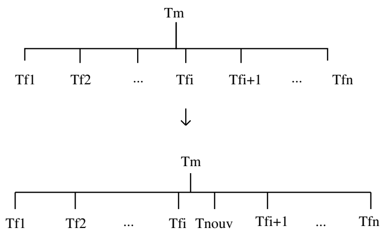


Ajout d'un fils Tnouv au rang i dans la tâche séquentielle Tm ayant pour fils initiaux Tf1, Tf2,...,Tfn

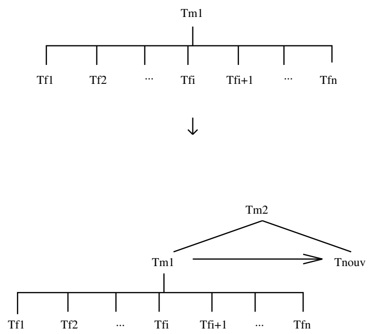


Transformation de la tâche séquentielle Tm1 en une tâche de choix, avec comme choix supplémentaire la tâche Tnouv et introduction d'une tâche fictive Tm2

Opérateurs élémentaires sur une tâche de choix

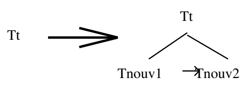


Ajout d'un choix supplémentaire Tnouv à la tâche de choix Tm

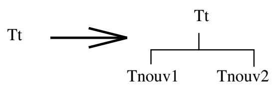


Ajout d'une séquence Tnouv après la tâche mère Tm1 avec introduction d'une nouvelle tâche fictive Tm2

Opérateurs élémentaires sur une tâche terminale



Transformation d'une tâche terminale Tt en une tâche séquentielle. Deux nouvelles tâches Tnouv1 et Tnouv2 sont introduites.



Transformation d'une tâche terminale Tt en une tâche de choix. Deux nouvelles tâches Tnouv1 et Tnouv2 sont introduites.

Fig. 6.2 - : Opérateurs élémentaires

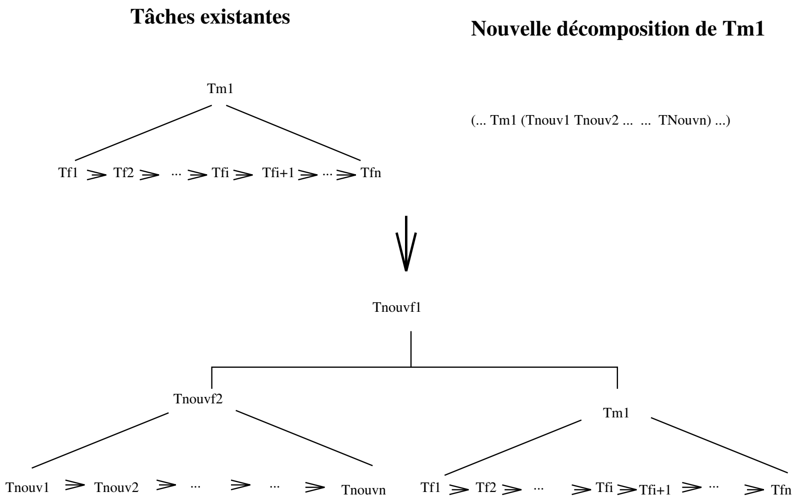


FIG. 6.3 - : Premier type d'opérateur avec plusieurs nouvelles tâches. Dans cette figure, deux tâches fictives ont été ajoutée: Tnouvf1 et Tnouvf2 ainsi que les nouvelles tâches définies par l'utilisateur (Tnouv1, Tnounv2...Tnouvn).

Chapitre 7

L'électromyographie, un domaine d'application possible

Bien que sujet de recherche théorique, le raisonnement à partir de cas est surtout destiné à améliorer la résolution de problèmes dans un domaine bien particulier et notamment à gagner une grande efficacité. Dans ANAÏS, nous avons tout d'abord recherché une mesure de similarité structurelle adaptée à un environnement de résolution de problèmes par décompositions successives. Cet environnement sert de plate-forme pour l'élaboration de bases de connaissances et notamment pour une base de connaissances en électromyographie.

Après une brève présentation de l'électromyographie, nous présenterons notre travail préliminaire dans le domaine, puis nous décrirons le fonctionnement d'ANAÏS à travers un exemple. Nous discuterons ensuite les connaissances prises en compte et celles, trop complexes, difficilement intégrées.

7.1 Présentation de l'électromyographie

L'électromyographie est un ensemble de techniques permettant de préciser le diagnostic de maladies neuro-musculaires. Ces techniques reposent sur l'étude des réactions musculaires et nerveuses, suite à des stimulations électriques par des aiguilles intra-musculaires. Trois types différents de maladie sont possibles : atteinte du système nerveux, atteinte du système musculaire et atteinte de la jonction neuro-musculaire. Les problèmes les plus complexes sont de choisir les tests à effectuer (ceux-ci doivent en effet être réduits car ils sont très douloureux) et d'évaluer les résultats de ces tests par rapport aux hypothèses de départ. L'examen électromyographique est complexe car il relève plus d'une centaine de valeurs et plus d'une centaine de diagnostics sont possibles. D'autre part, il existe plus de 4000 examens différents et il convient de bien choisir la suite des examens à effectuer de manière à ce que l'ensemble des examens ne soit pas trop lourd mais tout de même suffisant pour envisager toutes les pathologies possibles [ZVC92].

7.2 Difficultés liées au domaine

Pour faire face à ces nombreuses difficultés, plusieurs systèmes informatiques ont été développés. Des systèmes de traitement du signal permettent d'analyser les valeurs numériques en les regroupant de manière synthétique dans un tableau. Des systèmes à base de connaissances permettent de formuler un diagnostic. KANDID [FFRV89], ADELE [RCBV87] et NEUROP [RVB91] sont des systèmes à base de règles de production et MUNIN [AJA+89] est un système bayésien basé sur les probabilités. Ces systèmes apparaissent insuffisants car ils ne prennent pas en compte la globalité du processus électromyographique.

Le système MYOSIS [ZVC92], développé au sein de l'équipe, permet la prise en compte de toutes les étapes du diagnostic. Il s'avère cependant que le système se comporte comme un novice en effectuant tous les examens possibles pour chaque maladie évoquée. En dotant le système d'un module de raisonnement par cas, il est possible, d'une part d'optimiser le protocole d'examens à effectuer et d'autre part, de guider le médecin électromyographiste dans sa recherche de pathologies associées.

Dans le projet actuel au sein de l'équipe, les objectifs à atteindre à court terme et à long terme sont : la constitution d'un modèle de connaissances pour décrire les cas répertoriés, la définition des mécanismes d'inférence pour aider la résolution d'un nouveau cas, doter le système de capacité d'apprentissage pour améliorer la gestion de la base de cas.

Dans ANAÏS, nous nous proposons de modéliser et d'indexer les cas électromyographiques à l'aide de SHIRKA et de SCAI (voir 3.4 et 4), d'aider l'utilisateur dans la résolution de son problème en lui proposant des cas analogues et enfin, à plus long terme, d'enrichir incrémentalement la base de tâches afin de faciliter l'acquisition de nouvelles connaissances.

7.3 Connaissances déjà acquises dans le projet

Un premier travail de recueil de connaissances a été effectué par J.Willamowski dans le cadre d'un stage de DESS [Wil90]. Le but principal en était de modéliser le raisonnement dans le domaine de l'électromyographie afin d'apporter une aide à la décision de diagnostic et de faire un tuteur intelligent. Le système a été réalisé dans une architecture « blackboard » avec l'outil TRAM [Koe89]. Les différentes étapes du raisonnement ont été répertoriées : une première étape se charge d'acquérir les données cliniques (symptômes, facteurs de risque et facteurs limitatifs), il faut ensuite évoquer des hypothèses générales qui seront raffinées au cours de l'examen, l'examen proprement dit constitue la troisième étape, puis vient en dernier l'interprétation finale qui doit fournir un bilan de tous les tests effectués. Ces différentes étapes sont redécomposées, ainsi par exemple, l'examen électromyographique est découpé en quatre parties : l'évocation et le choix des hypothèses, le choix du protocole de test à effectuer, l'exécution du protocole et l'évaluation du protocole. Différents niveaux d'abstraction de la connaissance ont été repérés, il s'agit principalement d'une conclusion électromyographique au niveau d'un ou plusieurs segments de nerfs ou muscles, au niveau d'un nerf ou d'un muscle, au niveau d'une région particulière et enfin au niveau du corps entier du patient.

7.4 Analyse du travail relatif à l'application d'ANAÏS dans le domaine de l'électromyographie

Supposons que la conclusion de l'interprétation de certains tests soit normale et que l'utilisateur ne sache pas quels sont les prochains tests à faire. Ceux-ci dépendent du profil du patient, de ses antécédents cliniques ainsi que de restrictions (malade diabétique, âgé, allergique...). Après avoir décrit l'enchaînement de ses actions, il ne sait pas s'il faut changer de protocole de tests, s'il faut restreindre le protocole en cours ou au contraire l'étendre. ANAÏS essaie alors de retrouver les cas ayant le maximum de résolutions en commun avec le problème décrit. Pour ce faire, les caractéristiques du cas en cours de résolution sont demandées (voir 3.4).

La base de connaissances du système Myosis est écrite en SHIRKA et les cas électromyographiques sont représentés à l'aide du formalisme de SCAI. Il a donc fallu dégager des tâches de l'ensemble des inférences nécessaires dans le traitement d'un cas dans MYOSIS (voir figure 7.1). Malgré les difficultés rencontrées lors de l'acquisition de cas électromyographiques (voir section 7.5), une première base de tâches électromyographiques a été élaborée. Cette base rend compte des tâches générales et de leur enchaînement nécessaire pour résoudre des cas. Elle est cependant très insuffisante, car elle ne prend encore en compte les diagnostics intermédiaires, effectués sur les différents segments de muscles et de nerfs. D'autre part, la communauté électromyographique est actuellement en train d'élaborer des cas « consensus» (c'est-à-dire sur lesquels tout le monde est d'accord quant aux différents diagnostics) et seuls cinq ou six cas sont considérés comme des cas en « or».

La base de tâches incomplète devra être complétée au fur et à mesure de l'acquisition de nouveaux cas et nous tentons de définir une procédure automatique pour modifier incrémentalement la base de tâches afin qu'elle couvre de plus en plus de cas (voir chapitre 8).

D'autres exemples acquis dans un autre projet concernant une exploitation connexionniste des connaissances [Iodanova 90] ont été utilisés pour structurer et compléter les connaissances électromyographiques.

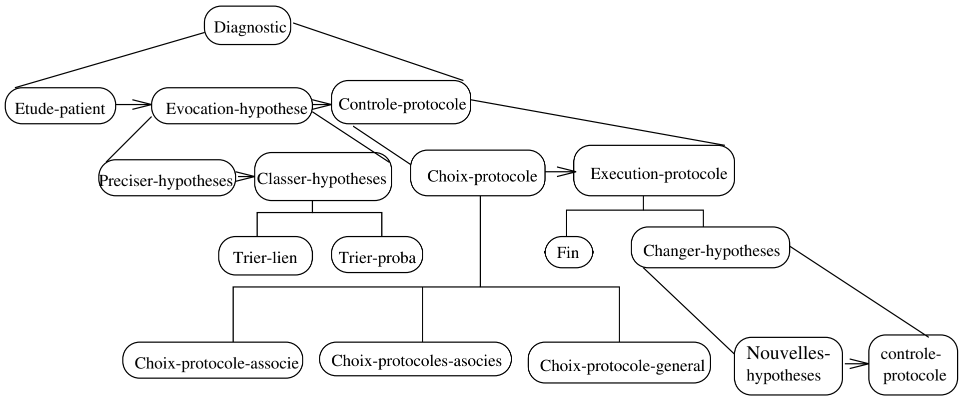


FIG. 7.1 - : Base de tâches en électromyographie

Les caractéristiques dans ce domaine sont principalement les symptômes observés et les facteurs limitatifs inhérents à chaque patient. Lors de la résolution d'un diagnostic, le spécialiste peut à l'aide de SCAI, instancier le réseau de tâches en indiquant ses choix.

7.5 Exemple électromyographique

7.5.1 Description générale par l'expert

La patiente a 64 ans (facteur de risque), l'âge influe sur la vitesse de conduction nerveuse. Le poids (68kg pour 1m58) est aussi un facteur de risque. La taille influe sur l'évaluation des résultats (le temps de conduction est différent). Le temps d'examen donne une indication quant à la complexité du problème. Le nom de l'examinateur permet de savoir s'il s'agit d'un expert ou d'un novice. Cette différence est en effet essentielle vis-à-vis du choix du protocole (qui est souvent très différent entre un expert et un novice), vis-à-vis de l'évaluation et du diagnostic final.

L'hérédité joue aussi un rôle important, c'est en effet un facteur de risque qui influe sur le choix des protocoles et sur le diagnostic final. La prise de médicaments à risque (ici thérapie stéroïde) est aussi un facteur de risque. L'hérédité, les symptômes, la prise de médicaments, les pathologies connues influent sur l'évocation d'hypothèses et donc sur le choix de protocoles. La raison pour laquelle on envoie le patient (polymyosite dans cet exemple) influe très fortement sur l'évocation d'hypothèses. Cette indication est normalement bien posée mais il se peut qu'il y ait parfois révocation d'hypothèses. Deux diagnostics sont en réalité établis, le diagnostic EMG (ElectroMyoGraphique) : myopathie moyenne et le diagnostic final ; polymyosite chronique possible.

Les tests effectués sont essentiellement à stimulation répétitive (troubles de la jonction neuro-musculaire). Les tests sur les nerfs sont effectués de manière à exclure l'hypothèse neuropathie (seuls quelques tests significatifs sont effectués). Les tests sur les muscles sont ceux des protocoles de la myosite et de la myopathie. Il est important de noter que le diagnostic EMG peut être différent pour un novice et un expert, les tests sont d'ailleurs souvent différents.

7.5.2 Facteurs pris en compte dans ANAÏS

Dans les tâches SCAI électromyographiques définies, les connaissances essentielles et intervenant de manière significative sont prises en compte. Notamment les informations suivantes :

- Facteurs de risque : âge, poids, thérapie stéroïde.
- Analyses médicales : biopsy musculaire.Hypothèses évoquées : myopathie.
- Symptômes : myalgie (douleurs dans les muscles). Hypothèses évoquées : Myosite forte.
- Choix des protocoles lié à des hypothèses diffuses : protocole associé à la neuropathie (diagnostic différentiel), protocole associé à la myopathie, protocole associé à la myosite.
- Conclusion EMG : myopathie moyenne, polymyosite chronique possible.

Toutes les connaissances prises en compte sont renseignées dans la définition des tâches elle-même, ainsi que dans les entrées et sorties de celles-ci.

7.5.3 Facteurs non pris en compte dans ANAÏS

Étant donnée la complexité du domaine, l'incomplétude de la base de connaissances de MYOSIS et les contraintes de l'environnement de résolution de problèmes, toutes les connaissances relatives aux cas recueillis auprès des experts n'ont pu être prises en compte. D'autre part, certaines informations n'influencent que très peu le raisonnement suivi par les médecins électromyographistes.

Le nom de l'examinateur (pour savoir s'il s'agit d'un novice ou d'un expert) n'est pas pris en compte. On ne sait pas actuellement comment réferrer cette différence. La raison pour laquelle le patient subit un examen n'est pas prise en compte. Ceci influe pourtant fortement sur le choix de l'hypothèse. En principe, après examen du patient, les hypothèses évoquées viennent soit confirmer l'hypothèse d'envoi, soit la remplacer.

Les résultats des tests eux-mêmes et l'influence sur le choix de la conclusion. Chaque évaluation influe en elle-même peu sur la conclusion (sauf certains tests significatifs). Dans le choix des tests, quelque soient les hypothèses évoquées, il y a toujours des tests sur les nerfs et sur les muscles. Le pourcentage de tests sur les nerfs est plus élevé lorsqu'il s'agit d'hypothèses de neuropathie. Cependant, dans le cas d'hypothèses de maladie myopathie, quelques tests sur les nerfs sont effectués pour exclure l'hypothèse neuropathie. Quelques résultats de tests peuvent modifier l'hypothèse évoquée et il est alors possible d'arrêter le protocole en cours et le modifier. Dans ANAÏS, pour simplifier, on a supposé que tout le protocole était effectué, puis que était prise soit une conclusion , soit une nouvelle hypothèse engendrant de nouveaux protocoles (complétion du protocole précédent).

7.6 Description du fonctionnement d'ANAÏS sur un exemple électromyographique

Après le chargement la base de tâches électromyographiques, quelques problèmes ont été résolus pour constituer la base de cas. Le cas1 est décrit dans la figure 7.2.

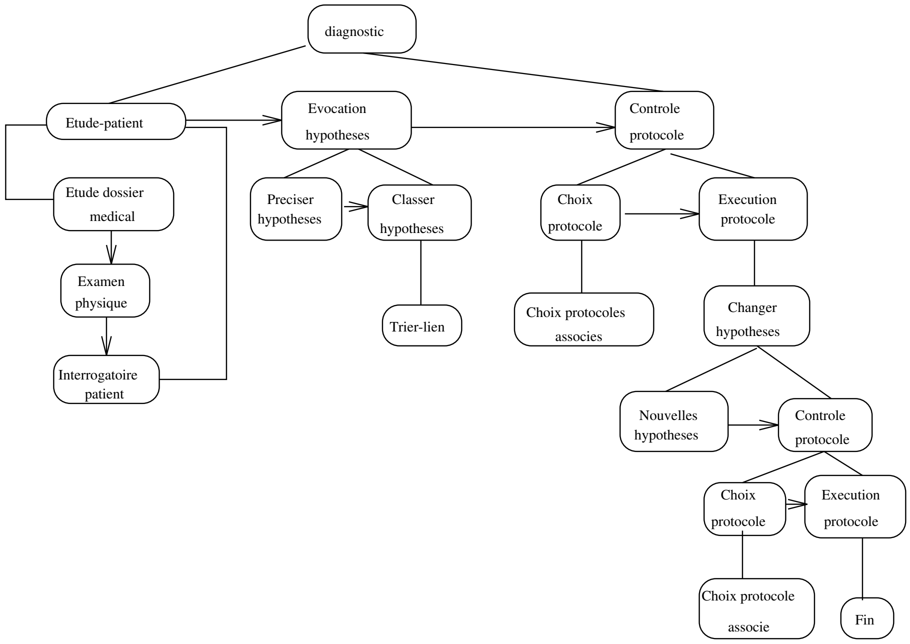


FIG. 7.2 - : Les tâches du cas 1

La trace des tâches dans le cas1 est la suivante :

```
(cas1 (diagnostic
         (etude-patient
            (etude-dossier-medical examen-physique interrogatoire-patient)
          evocation-hypotheses (preciser-hypotheses classer-hypotheses (trier-lien))
          controle-protocole
    (choix-protocole (choix-protocoles-associes)
             execution-protocole
       (changer-hypotheses
                 (nouvelles-hypothese controle-protocole
                    (choix-protocole (choix-protocole-associe)
                     execution-protocole (fin)))))))).
```

Supposons les autres cas suivants dans la base :

```
(cas2 (diagnostic
(etude-patient (etude-dossier-medical examen-physique interrogatoire-patient)
         evocation-hypotheses (preciser-hypotheses classer-hypotheses (trier-proba))
 controle-protocole (choix-protocole (choix-protocole-general)
                             execution-protocole (fin)))))
```

et

```
(cas3 (diagnostic
(etude-patient (etude-dossier-medical examen-physique interrogatoire-patient)
         evocation-hypotheses (preciser-hypotheses classer-hypotheses (trier-proba))
         controle-protocole (choix-protocole (choix-protocole-associe)
                             execution-protocole (fin)))))
```

L'utilisateur essaie ensuite de résoudre un nouveau problème et exprime des tâches utiles :

```
(RP1 (classer-hypotheses (trier-lien))).
(RP2 (choix-protocole-associe)).
(RP3 (execution-protocole (changer-hypotheses (nouvelles-hypothese)))).
```


```
(RP1 (classer-hypotheses (trier-lien))).
(RP2 (choix-protocole-associe)).
(RP3 (execution-protocole (changer-hypotheses (nouvelles-hypothese)))).
```


Dans le domaine électromyographique, il est important de connaître le nombre d'appels de la tâche controle-protocole. Le raisonnement global du cas électromyographique dépend fortement de ce nombre d'appels. Dans le cas1, la tâche controle-protocole est rappelée une seconde fois, ce qui signifie que les tests des protocoles effectués soit, n'ont pas été suffisants pour déterminer un diagnostic, soit, ont infirmé une hypothèse évoquée ou encore ont permis de détecter des anomalies évoquant des maladies supplémentaires. Ceci implique donc qu'une nouvelle hypothèse soit évoquée afin de générer d'autres protocoles qui permettront d'élaborer un autre diagnostic électromyographique. Tenir compte du nombre d'appels de la tache controle-protocole est donc primordial dans le domaine électromyographique.

La recherche des tâches communes pour les exemples précédents donne les résultats suivants :

- Entre cas1 et RP1 : (classer-hypotheses (trier-lien)).
- Entre cas1 et RP2 : (choix-protocole-associe).
- Entre cas1 et RP3 : (execution-protocole (changer-hypotheses (nouvelles-hypothese))).
- Entre cas2 et RP1 : (classer-hypotheses).
- Entre cas2 et RP2 : ().
- Entre cas2 et RP3 : (execution-protocole).
- Entre cas3 et RP1 : (classer-hypotheses).
- Entre cas3 et RP2 : (choix-protocole-associe).
- Entre cas3 et RP3 : (execution-protocole)

Le calcul des importances des tâches donne le résultat référencé dans la figure 7.3

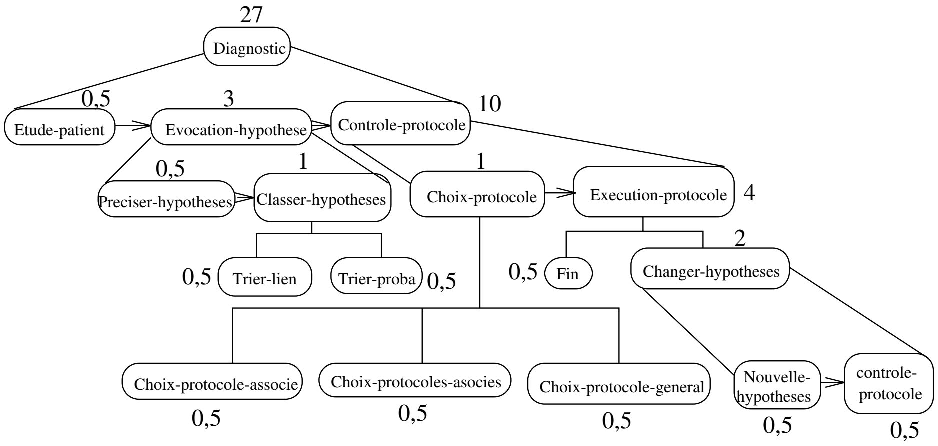


FIG. 7.3 - : Valeur finale du poids du nœud

Les calculs de similarité :
$Sim_{struct}(cas1, RP1) = 1.5 \times 1.5 \div 57 = 0.039$
$Sim_{struct}(cas1, RP2) = 0.5 \times 0.5 \div 57 = 0.004$
$Sim_{struct}(cas1, RP3) = 6.5 \times 6.5 \div 57 = 0.741$
$Sim_{struct}(cas2, RP1) = 1 \times 1.5 \div 48.5 = 0.030$
$Sim_{struct}(cas2, RP2) = 0 \times 0.5 \div 48.5 = 0$
$Sim_{struct}(cas2, RP3) = 4 \times 6.5 \div 48.5 = 0.536$
$Sim_{struct}(cas3, RP1) = 1 \times 1.5 \div 48.5 = 0.030$
$Sim_{struct}(cas3, RP2) = 0.5 \times 0.5 \div 48.5 = 0.005$
$Sim_{struct}(cas3, RP3) = 4 \times 6.5 \div 48.5 = 0.536$

Si RP est l'ensemble des RPi, les résultats finaux sont les suivants : $Sim_{struct}(cas1, RP) = 0.784$, $Sim_{struct}(cas2, RP) = 0.566$ et $Sim_{struct}(cas3, RP) = 0.571$.
Le cas1 est donc celui le plus proche du problème en cours de résolution, il est donc proposé à l'utilisateur. Il reste donc à compléter ce problème à résoudre, en définissant de nouvelles tâches si nécessaire et en réutilisant des tâches du cas1.

7.7 Conclusion

Le traitement des cas électromyographiques est décrit à travers les exemples illustrant chaque partie d'ANAÏS. La base de connaissances de MYOSIS est encore incomplète, bien qu'elle possède déjà plusieurs centaines de classes. Notamment, tous les tests existants ne sont pas encore répertoriés ainsi que les protocoles associés aux hypothèses évoquées.

Près de 64 cas différents sont stockés sous la forme d'un fichier, mais le raisonnement associé à ces cas n'est pas encore acquis. Lors des interviews avec les médecins, nous avons pu mesurer la difficulté d'expression du raisonnement suivi. Cependant, cela leur a apporté un éclairage nouveau sur leur manière de faire et des conflits entre médecins sont apparus. La plus grande difficulté a donc été de repérer les tâches inhérentes aux raisonnements des médecins. Nous espérons compléter la base de tâches actuelle en utilisant ANAÏS à l'aide des 60 autres cas consensus existants.

Dans l'évaluation de la similarité entre cas, nous avons constaté que des cas présentant au départ des connaissances très différentes (symptômes, signes et historiques différents) se ressemblent dans la démarche suivie. La similarité obtenue n'est donc pas due au fait que les conclusions ou les entrées des cas soient identiques. Pour affiner cette similarité dans le domaine électromyographique, il faudrait à partir des cas présentant des démarches semblables, les réévaluer par rapport à leurs données. Cela signifie qu'après l'évaluation d'ANAÏS vis-à-vis des schémas de raisonnements des cas, qu'il faut tenir compte des valeurs des entrées/sorties de toutes les tâches communes aux cas similaires. Ceci sera très certainement valable dans d'autres domaines d'expertise. Une première amélioration d'ANAÏS est donc d'élaborer une troisième étape, après l'étape d'extraction et l'étape de sélection, pour affiner la dernière sélection.

Chapitre 8

Conclusion et perspectives

Afin de faciliter la résolution de problèmes dans un contexte où la base de connaissances est incomplète, nous avons proposé de réutiliser l'ensemble des étapes de résolution d'anciens problèmes. Bien que la problématique de réutilisation de problèmes anciennement résolus soit inhérente à la discipline du RBC, notre approche se démarque par le fait que la mémoration des cas est fondée sur une similarité entre plusieurs résolutions de cas.

Deux objectifs principaux sous-tendent notre démarche : une aide à la résolution de problèmes et un enrichissement incrémental de la base de connaissances. En parallèle à ces deux objectifs, deux hypothèses nous ont guidé : il existe une base de tâches du domaine incomplète et l'utilisateur peut exprimer une partie de la résolution de son problème à l'aide des tâches existantes.

Pour aider l'utilisateur dans la phase de résolution de problèmes, des résolutions de problèmes similaires, sous forme de décomposition en tâches et sous-tâches, sont proposées. Ainsi, la résolution de problèmes peut être effectuée par deux méthodes différentes : par instanciation de plans généraux et par extraction de cas similaires. Ceci permet aussi de faciliter l'acquisition des connaissances puisque l'utilisateur a à sa disposition des cas similaires et qu'il peut les combiner et les compléter pour résoudre de nouveaux problèmes.

Cependant, l'automatisation de la complétion de la base de connaissances par rapport à la résolution de nouveaux problèmes est très souhaitable. Dans notre contexte, il s'agit donc de modifier incrémentalement la base de tâches du domaine de manière à ce qu'elle puisse résoudre tous les nouveaux problèmes proposés. Des opérateurs de base ont été proposés ainsi que des cas « simples » où l'algorithme de modification de la base de tâches est élémentaire. L'étude de la modification de la base de tâches, lorsque les nouveaux cas à insérer modifieraient la structure même de la décomposition des tâches (parties de la base de tâches à déplacer), demande à être approfondie. Celle-ci serait à même de découvrir des critères (automatiques, demandant une intervention de l'utilisateur ou encore nécessitant des méta-connaissances sur le domaine) permettant de choisir les modifications de structure à apporter.

Par rapport aux travaux actuellement opérationnels, plusieurs travaux techniques sont indispensables. D'une part, du point de vue ergonomique, il est indispensable de proposer une interface graphique permettant à l'utilisateur de : couper/coller les tâches des cas proposés, créer de nouvelles tâches et les insérer dans la base existante afin de construire graphiquement la résolution du problème. Cette interface n'a pas été construite car sa réalisation implique la génération d'un compilateur traduisant le langage graphique dans le langage SCAI. Un tel compilateur est assez complexe à réaliser et demanderait à lui seul, une étude approfondie.

D'autre part, dans l'application électromyographique, il est nécessaire d'affiner la base de tâches actuelle. Mais nous avons justement constaté qu'il était beaucoup plus simple pour les experts, d'exprimer la résolution de nouveaux cas que de trouver des nouvelles tâches au modèle. Il est donc naturel d'étudier le problème de la modification incrémentale automatique d'une base de tâches, en fonction des nouveaux cas qu'il faudra couvrir.

Dans cette perspective, nous avons dégagé les opérations élémentaires de modification d'une base de tâches (voir section 6.4). Quelques cas simples qui n'engendrent pas différentes possibilités d'application de ces opérations ont été dégagés.

D'autres perspectives à plus longue échéance s'inscrivent dans la continuité de ce travail. Poursuivant l'objectif de faciliter l'acquisition des connaissances, il est intéressant d'initialiser une base de tâches. Ainsi, à partir d'un ensemble de cas résolus, un algorithme de généralisation peut être mis en œuvre pour calculer des tâches plus générales. Si les différents cas résolus sont issus de plusieurs bases de tâches différentes, nous touchons alors au domaine de l'analogie. Dans ce contexte, des travaux débutent [Col94] sur la recherche de similarité entre base de tâches issues de domaines différents.

Bibliographie

[AJA+89] S. Andreasen, F.V Jensen, S.K Andersen, B Falck, Kjoerulff, A.R Sorensen M. Woldbye, and A. Rosenfalck. Munin - an expert emg assistant. *Computer-Aided Electromyography and expert-systems*, 1989. Desmedt, J.E editor.

[Alt88] Richard Alterman. An adaptative planner. In *DARPA Workshop on Case-Based Reasoning*. Morgan Kaufman, 1988.

[AR88] K.D Ashley and E.L Rissland. Credit assignment and the problem of competing factors in case-based reasoning. In *DARPA Workshop on Case-Based Reasoning*, 1988.

[Aus89] Nathalie Aussenac. Conception d'une méthodologie et d'un outil d'acquisition de connaissances expertes. Thèse d'université, Université Paul Sabatier de Toulouse, October 6 1989.

[Bai86] William Bain. Judge : A case-based reasoning system. In Jaime G. Carbonell Ryszard S. Michlaski, editor, *A guide to Current Research*, pages 1–4. Kluwer Academic publishers, 1986.

[Bea93] Nathalie Beauboucher. Anais : Un système de raisonnement à partir de cas dans un environnement de résolution de problèmes. In *Journées VOLCAN-IA 93*, pages 245–260, Clermont-Ferrand (France), Mars 1993.

[Bis93] Gilles Bisson. Induction de bases de connaissances en logique des prédicats. Thèse d'université, Université d'ORSAY, April 30 1993.

[Bra89] L.K Branting. Integrating generalizations with exemplar-based reasoning. In *DARPA Workshop on Case-Based Reasoning*, Holiday Inn, Florida, 1989. Morgan Kaufman.

[BS91] Isabelle Bichindaritz and Brigitte Seroussi. Validité a priori des inférences produites par un raisonnement à partir de cas : objectivation de l'analogue par des critères de ressemblance fonctionnellement quantifiés. *Huitième congrès reconnaissance des formes et intelligence artificielle*, 3:1151–1158, 1991. ISBN 2 - 90 36 77 103 - 8.

[Cal90] Sergio Calabretta. Acknowledge : Vers l'amélioration du processus d'acquisition des connaissances. *Actes des journées sur l'acquisition des connaissances*, April 1990.

[Cap93] Cécile Capponi. Classification des classes par les types. In *2nd journées représentation par objets (RPO)*, pages 215–224, La Grande Motte (France), June 1993.

[Car83] J. Carbonell. *Learning by analogy : formulating and generalizing plans from past*, volume 1. Morgan Kaufmann, 1983.

[Car86] Jaime G. Carbonell. Derivational analogy : a theory of reconstructive problem solving and expertise acquisition. In *Machine Learning, An artificial intelligence approach*, pages 371–391. Morgan Kaufman, 1986. Vol 2.

[Cer89] José Luis Aguirre Cervantes. Construction automatique de taxonomies à partir d'exemples dans un modèle de connaissances par objets. Thèse d'université, Institut National Polytechnique de Grenoble, July 6 1989.

[Cha83] B. Chandrasekaran. Towards a taxonomy of problem solving types. *Artificial Intelligence*, 4(1):9–17, 1983.

[Cha87] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *International Joint Conference on Artificial Intelligence*, volume 2, pages 1183–1192, Milan, Italy, 1987.

[CHL+91] François Charpillet, Marie-Christine Haton, Brigitte Hassan Lâasri, Pierre Marquis, Thierry Mondot, Amedeo Napoli, Jean-Paul Haton, and Nadjet Bouzid. *Le raisonnement en intelligence artificielle, modèles, techniques et architectures pour les systèmes à base de connaissances.* InterEditions, 1991.

[Cla84] W.J Clancey. Classification problem solving. In *International Joint Conference on Artificial Intelligence*, pages 49–55, Austin, Texas, 1984.

[Cla85] W.J Clancey. Heuristic classification. *Artificial Intelligence*, 27:215–251, 1985.

[Col94] Elsa Collin. Détection de similarités de raisonnement représentés par des tâches. Rapport de d.e.a, LIFIA, 1994. A paraître.

[Coo91] D.J. Cook. The base selection task in analogical planning. In *IJCAI*, pages 790–795, Sydney, Australia, 1991.

[D'e91] Marie-Christine D'erceville. Le projet KASA ou KADS au service de l'apprentissage. In *Journées d'acquistion des connaissances*, 1991.

[Der86] Nachum Dershowitz. Programming by analogy. In *Machine Learning, An artificial intelligence approach*, pages 393–421. Morgan Kaufman, 1986. Vol 2.

[DLPT82] Edwin Diday, Jacques Lemaire, Jean Pouget, and Françoise Testu. *Éléments d'analyse de données.* Dunod, 1982.

[Don89] Dan Donahue. Ogre : Generic reasoning from experience. In *DARPA Workshop on Case-Based Reasoning*, pages 248–252, Holiday Inn, Florida, 1989. Morgan Kaufman.

[Doy79] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[FFRV89] A. Fuglsang-Frederiksen, J. Ronager, and S. Vingtof. Pc-kandid, an expert system for electromyography. *Artificial Intelligence in Medecine*, 1, 1989.

[Fis87] D. Fisher. *Knowledge acquisition via incremental conceptual clustering*, volume 2, pages 139–172. Tioga, Palo Alto, Calif., r.s michalski and j. carbonell and t. mitchell edition, 1987.

[Gan85] Jean-Gabriel Ganascia. La conception des systèmes experts. *La recherche*, 17:1143–1151, octobre 1985.

[GC85] M. Gluck and J. Corter. Information, uncertainty and the utilitie of categories. In *Proceedings of the seventh annual conference of the cognitive Science society*, pages 283–287, Irvine, CA, 1985.

[Gen83] D. Gentner. Structure mapping: A theorical framework for analogy. In *Cognitive Science*, pages 155–170. Ablex publishing corporation, 1983.

[GLF89] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[GW88] J. Breukerand G.Schreiber and B. Wielinga. modelling in kbs development. In *Huitièmes journées internationales sur les systèmes experts et leurs applications*, Avignon, 1988.

[HJB92] Lawrence B. Holder, Diane J.Cook, and Horst Bunke. Fuzzy substructure discovery. In *Proceedings of Machine Learning*, pages 218–223. Morgan-Kaufmann, 1992.

[HK87] P. Harmon and D. King. *Systèmes experts professionnels. Conception et implémentation.* Collection MIM-IA.Editions Masson., 1987.

[HKL+89] Franck R. Hickman, Jonathan L. Killin, Lise Lnad, Tim Mulhall, David Porter, and Robert M. Taylor. *Analysis For Knowledge-Based Systems : A Practical Guide to the KADS Methodology.* Ellis Horwood Limited, Chichester, 1989. ISBN 0-7458-0689-9.

[HMH92] Habib Hadj-Mabrouk and Bernard Houriez. Acquisition de connaissances et apprentissage pour l'élaboration d'une base de connaissances. In *Actes des premières journées Francophones d'apprentissage et d'explicitation des connaissances*, pages 29–46, Dourdan, April 15-17 1992.

[HRWL83] Hayes-Roth, Waterman, and Lenat. *Building Expert Systems.* Addison-Wesley, 1983.

[HW87] J. Hogg and S. Weiser. Otm : Applying objects to tasks. *OOPSLA 87 Proceedings*, pages 388–393, october 1987.

[KM92] Janet Kolodner and William Mark. Guest editor's introduction : Case-based reasoning. In *IEEE expert, Intelligent systems and their applications*, pages 5–6, october 1992.

[Kod92] Yves Kodratoff. Recent advances in machine learning. *International journal of Pattern Recognition and Artificial Intelligence*, 6(4):469–511, 1992.

[Koe89] Koenig. Tram : une architecture blackboard en robotique autonome. thèse de doctorat, INPG, Grenoble, 1989.

[Kol85] J.L Kolodner. Using experience in clinical problem solving. Technical report git-ics-85/21, Scholl of Information and Computer Science, 1985. Computer Science, Georgia Institut of Technology.

[Kol88] J.L Kolodner. Extending problem sover capabilities through case-based inference. In *Proceddings of the Case-Based Reasoning Workshop.* Morgan Kaufman, 1988.

[Kot88] Phyllis A. Koton. *Using Experience in Learning and Problem Solving*. PhD thesis, Departement of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 5 1988.

[Mar93] Olga Marino. Raisonnement classificatoire dans une représentation à objets multi-points de vue. Thèse d'université, Université Joseph Fourier, October 1993.

[Mic83] R.S Michalski. *A theory an Methodology of Inductive Learning*, pages 83–129. Tioga, Palo Alto, Calif., r.s michalski, j. carbonell, t. mitchell edition, 1983.

[Mic90] R. S. Michalski. *Machine Learning: An Artificial Intelligence Approach*, volume III, chapter III. Learning Flexible Concepts: Fundamental Ideas and Method Based on Two-Tiered Representation. Morgan Kaufmann, yves kodratoff, ryszard s. michalski edition, 1990.

[Mic93] R.S Michalski. Classification conceptuelle : Apprentissage à partir d'observations et classification d'objets struturés avec objectif. In *Apprentissage symbolique : une approche de l'Intelligence Artificielle*, volume 1, pages 245–294. Cépadues-Éditions, 1993.

[Mig92] Hervé Mignot. Etude d'une mesure de similarité pour le raisonnement par cas. Dea informatique, Université de Paris-Sud, société Isoft S.A, septembre 1992.

[MS83] R.S Michalski and R.E Stepp. *Learning from observation: Conceptual Clustering*, pages 331–363. Tioga, Palo Alto, Calif., r.s michalski, j. carbonell, t. mitchell edition, 1983.

[MWS+93] Claudine Médigue, Jutta Willamowski, Olivier Schmeltzer, Patrice Uvietta, François Rechenmann, François Chevenet, Guy Perrière, and C. Gautier. Modeling tasks for problem solving in molecular biology. In *Workshop IJCAI : Artificial Intelligence and Genome*, pages 67–76, Chambéry, 1993.

[Ned92] Claire Nedellec. How to specialize by theory refinment. In *ECAI*, pages 474–478, Vienna, August 1992.

[Nic87] Jacques Nicolas. Ally, un système logique pour la généralisation en apprentissage automatique. Thèse d'université, Université de Rennes I, Rennes, France, May 1987.

[NS91] Jean-Philippe Nault and Olivier Schmeltzer. La reconnaissance d'intentions par appariement d'objets. Rapport de projet, année spéciale intelligence artificielle, Institut Polytechnique de Grenoble, ENSIMAG, Juin 1991. Laboratoire ARTEMIS-IMAG.

[OGA92] Bruno Orsier, Arnaud Giacometti, and Bernard Amy. Application of neural networks to AI: Hybrid systems and cognitive models. In *AAAI workshop "Integrating Neural and Symbolic Processes The Cognitive Dimension*, San Jose, California, USA, July 1992.

[PGD91] Christine Pierret-Golbreich and Isabelle Delouis. *Task Centered Representation for Expert Sytems at the Knowledge Level*, pages 149–159. Springer-Verlad, lucs steels, barbara smith edition, 1991.

[Pin87] Suzanne Pinson. Méta-modèle et heuristiques de jugement : Le système credex. application à l'évaluation du risque crédit entreprise. Thèse d'université, Université de Paris VI, 1987. Thèse de doctorat.

[PR89] Marc Prevosto and François Rechenmann. Présentation et démonstration du logiciel SAID d'aide au traitement du signal. In *Actes journées de clôture des pré-études C2A*, CNET, Issy-les-Moulineaux (France), 1989.

[PR91] Thierry Poncabarré and François Rechenmann. SCAI: un environnement de développement de systèmes à base de connaissances en calcul scientifique et technique. In *Convention intelligence artificielle*, Paris, january 1991.

[RCBV87] Chantal Reynaud, Marie-Odile Cordier, A. Blinowska, and J. Verroust. Adèle- un système expert en électromyographie. In *IA-SANTE 87, SITEF, Pôle GBM*, Toulouse, Faculté de médecine, 1987.

[Rec85] François Rechenmann. Shirka : Mécanismes d'inférence sur une base de connaissances centrée objets. In *5 ième Congrès reconnaisances des formes et intelligence artificielle*, Grenoble, novembre 1985.

[Rou88] B. Rousseau. Vers un environnement de résolution de problèmes en biométrie. apport des techniques de l'intelligence artificielle et de l'interaction graphique. Thèse d'université, Université Claude Bernard, Lyon I, Février 1988.

[RS89a] Christopher Riesbeck and Roger Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, The institute for the learning sciences Northwestern university, Evanston, Illinois, 1989.

[RS89b] E.L Rissland and D.B Skalak. Combining case-based reasoning and rule-based reasoning: A heuristic approach. In *International Joint Conference on Artificial Intelligence*, pages 524–530, Detroit, MI, 1989.

[RVB91] V. Rialle, A. Vila, and Y. Besnard. Heterogeneous knowledge representation using a finite automation and first order logic : a case study in electromyography. *Artificial Intelligence in Medecine*, 3, 1991.

[SB93] Claude Sammut and Ranan B.Barnerji. Apprendre des concepts en posant des questions. In *Apprentissage symbolique, une approche de l'Intelligence Artificielle*, pages 399–423. R.S Michalski and J.G Carbonell and T.M Mitchell and Y. Kodratoff, cépadues éditions edition, 1993. Traduction de Pierre Maquis.

[Sch82] Roger C. Schank. *Dynamic Memory, A theory of reminding and learning in computers and people.* Cambridge University Press, 1982.

[Sei89] C.M. Seifert. Analogy and case-based reasoning. In *Workshop on Case-Based Reasoning*, pages 125–129, Holiday Inn, Florida, 1989.

[SF86] J. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of AAAI-86*, pages 496–501, Philadelphia, PA, 1986.

[Sim85] R.L. Simpson. A computer model of case-based reasoning in problem solving. Technical report no. git-ics/85/18/, School of Information and Computer Science. Georgia Institut of Technology. Atlanta, GA., 1985.

[Sla91] Stephen Slade. Case-based reasoning: A research paradigm. *AI Magazine*, 12, SPRING 1991. numero 1.

[SLSV84] R.F Smith, G.M.E Lafue, E. Schoen, and S.C Vestal. Declarative task description as a user-interface structuring mechanism. *IEEE Computer*, 17(9):29–37, september 1984.

[SN91a] Katia Sycara and D. Navinchandra. Index transformation techniques for facilitating use of multiple cases. In *12 th International Joint Conference on Artificial Intelligence*, pages 347–352, Darling Harbour, Sydney, Australia, August 24-30 1991.

[SN91b] Katia Sycara and D. Navinchandra. Influences: A thematic abstraction for creative of multiple cases. In *CBR from DARPA*, pages 133–144, Washington, 1991.

[SW86] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986.

[TK93] Gheorghe Tecuci and Yves Kodratoff. *Systèmes apprentis dans des domaines à théories incomplètes*, volume 2, pages 453–469. R.S Michalski, J.G Carbonell, T.M Mitchell, Y. Kodratoff, cépadues-éditions edition, 1993.

[Tve77] Amos Tversky. Features of similarity. *Psychological review*, 84:327–352, 1977. By the American Psychologicl Association.

[UWZ93] Patrice Uvietta, Jutta Willamoski, and Danièle Ziébelin. Task based modeling for problem solving strategies. In *Actes 5th international conference on tools with artificial intelligence*, Cambridge (MA US), November 1993. to be appear.

[VC91] Manuela Veloso and Jaime Carbonell. Variable-precision case retrieval in analogical problem solving. In *Workshop on Case-Based Reasoning*, pages 93–106, Holiday Inn, Florida, 1991.

[Vog88] Claude Vogel. *Génie cognitif.* Masson, 1988.

[Vog89] Claude Vogel. Kod : Une méthode inspirée par l'anthropologie. est-elle applicable au droit ? *Supplément de Lilliade*, 2(53), Octobre 1989.

[Wil90] J. Willamoswski. Modélisation du raisonnement en électromyographie. Rapport de dess intelligence artificielle, Université de Savoie, Chambéry, 1990.

[ZVC92] D. Ziébelin, A. Vila, and F. Cruypenninck. Automatic reasoning and explanation in knowledge based system for emg diagnotics. In *IX International congress of electromyography and clinical neurophysiology*, Jérusalem, Israel, 1992.