Vers une méthodologie de développement de SMA adaptés aux problèmes d'optimisation

Marianne Flouret, Bruno Mermet, Gaële Simon

LIH, Université du Havre
BP 540
76058 Le Havre Cedex

RÉSUMÉ. *Développer des systèmes multiagents s'avère être une tâche compliquée. Avoir une garantie sur le résultat obtenu l'est plus encore. Dans cet article, nous décrivons une méthodologie qui aide à cela. Cette méthodologie est dédiée aux problèmes d'optimisation qui peuvent être résolus en combinant des contraintes locales. Nous présentons également des outils supports que nous avons développés pour aider à mettre en oeuvre la méthodologie présentée et qui ont été utilisés pour une application à la coloration de graphes.*

ABSTRACT. *Developing multi-agent systems may be a rather difficult task. Having confidence in the result is still more difficult. In this article, we describe a methodology that helps in this task. This methodology is dedicated to global optimization problems that can be solved combining local constraints. To support this methodology, we developed CASE tools which are also presented and that were applied to solve a graph colouring problem.*

MOTS-CLÉS : *Système multiagent, méthodologie, coloration de graphe*

KEYWORDS: *multi-agent system, methodology, graph colouring*

1. Introduction

Il existe déjà de nombreuses méthodes pour développer des systèmes multiagents (MAS) [IGL 99]. Ces méthodes sont en général constituées d'un ensemble de modèles servant à spécifier le problème, mais l'aspect méthodologique permettant d'aider à leur mise en oeuvre sur des problèmes réels est souvent absent [WOO 00, BRA 97]. Dans cet article, nous présentons les bases d'une méthodologie, associée à une méthode, pour développer des systèmes multiagents dédiés à des problèmes d'optimisation. Des outils d'aide ont également été réalisés et utilisés pour une application à la coloration de graphe, mais cela n'est pas présenté.

2. Méthodologie

Le but de notre recherche est d'avoir une méthodologie, une méthode ainsi que des outils d'aide à la conception de SMA adaptés à la résolution de problèmes distribués.

Deux aspects importants de notre méthodologie sont les suivants :
– à chaque instant, le système peut être arrêté ;

– plus le temps passe, meilleure est la solution proposée par le système.

2.1. *Conditions d'utilisation*

La méthodologie que nous présentons est dédiée aux problèmes qui peuvent être exprimés en termes de contraintes locales (CL). Une condition d'utilisation plus restrictive est que la méthodologie est consacrée à des problèmes d'optimisation pour lesquels il existe une solution triviale (éventuellement très mauvaise).

2.2. *La méthodologie*

2.2.1. *Variant global*

La première étape est d'introduire un variant, une notion souvent utilisée pour prouver la terminaison d'algorithmes. Un variant est une variable définie sur une structure totalement ordonnée, qui doit décroître à chaque itération et qui possède une borne inférieure. Ces deux propriétés impliquent la terminaison des itérations.

2.2.2. *Décomposition locale*

La seconde étape est peut-être la plus difficile : le problème à résoudre doit être exprimé en terme de sous-problèmes locaux. Cela consiste à diviser la solution du problème en plusieurs parties, pas nécessairement indépendantes. Chaque partie est associée à un sous-problème local. La résolution de chacun d'eux doit aider à la résolution du problème global. Le cas idéal est celui de sous-problèmes dont la résolution est une condition nécessaire et suffisante pour résoudre le problème global. Cependant, ce n'est pas toujours le cas. Une autre possibilité est d'avoir des sous-problèmes dont la résolution fait simplement décroître le variant global.

2.2.3. *Agentification*

Le problème global étant décomposé, il faut alors déterminer les agents du système. Bien sûr, une première idée consiste à associer chaque problème local à un agent, mais ce n'est pas toujours possible pour la raison suivante.

Pour agentifier un problème, deux types de contraintes doivent être considérées :
– chaque sous-problème doit être associé à un agent ;

– chaque propriété (ou donnée) doit être associée à un agent.

Chaque agent ne perçoit qu'une partie de son environnement. De plus, un agent étant autonome, aucun autre agent ne peut modifier directement ses propriétés. Nous avons appelé ces deux contraintes le *principe de localité*. Ainsi, si la résolution de deux sous-problèmes nécessite de modifier une même propriété, affecter les deux sous-problèmes à deux agents différents est impossible.

La solution choisie consiste à changer la structure des sous-problèmes locaux afin que la modification d'une propriété ne puisse intervenir que dans la résolution d'un sous-problème. Ansi, chaque modification de propriété est contrôlée par un et un seul agent. Les autres agents ayant besoin de connaître la valeur de cette propriété doivent donc avoir son agent propriétaire dans leur réseau d'accointances et peuvent connaître sa valeur par échange de messages. Les sous-problèmes résultant de cette restructuration sont appelés Sous-Problèmes Orientés Propriétés (SPOP). Cette étape fournit en outre les relations d'accointance du SMA.

2.2.4. *Comportement des agents*

Nous considérons les agents comme réactifs et sociaux, ce qui signifie qu'ils peuvent réagir à des changements de leur environnement et communiquer avec d'autres agents.

Comportement général : Chaque SPOP est affecté à un agent. Ainsi, le comportement général d'un agent se résume donc à ne rien faire si son problème est résolu (l'agent est *satisfait*) ou bien essayer de trouver une solution à son problème dans le cas contraire.

Résoudre un problème : Pour le problème global, nous avions introduit un variant. Nous devons faire de-même pour chaque SPOP de façon à ce que la décroissance d'un variant local implique la non-croissance stricte du variant global. Chaque SPOP peut être divisé en sous-buts dont la résolution fait décroître le variant local. Ainsi, un agent insatisfait choisit un sous-but et doit le résoudre.

Quand un sous-but doit être résolu, il y a deux possiblités :
– ou bien il peut être résolu par l'agent qui choisit alors un nouveau sous-but ;

– ou bien l'agent ne peut le résoudre.

Il y a deux raisons pouvant expliquer qu'un sous-but ne peut être résolu :

– soit on est dans une situation de blocage : un autre agent empêche l'agent qui agit d'appliquer l'une de ses stratégies ;

– soit l'agent ne sait pas comment résoudre le sous-but dans la situation courante. Dans le deuxième cas, l'agent choisit un autre sous-but ou attend que la situation courante soit modifiée. Dans le premier cas, l'agent agresse l'agent gêneur. Ce comportement correspond à celui d'un éco-agent [DRO 93]. Le mécanisme d'agression est simulé par l'envoi d'un message d'agression. Un agent agressé doit fuir pour faire disparaître la situation de blocage, tout en préservant les contraintes locales CL. Lors de la fuite, le variant local peut être amené à augmenter. Si un agent ne peut pas fuir, il ignore l'attaque. Pour nous aider à spécifier le comportement des agents, nous avons utilisé le formalisme des automates à multiplicités [BER 01]. Ce formalisme peut aussi être utilisé pour d'autres types d'agents [MER 02].

3. Conclusion et perspectives

Cette méthode a été appliquée avec succèsà un problème de coloration de graphe. Il s'agissait d'illustrer la pertinence de cette méthodologie pour des problèmes réels. Les prolongements portent actuellement sur l'obtention d'une méthode complètement spécifiée par des modèles formels ou semi-formels (tels que, par exemple, les automates à multiplicités décrits ici) dédiés à l'aide au développement de SMA. Des outils de génie logiciel devront aussi être développés pour aider à appliquer aussi bien la méthodologie que la méthode présentées.

4. Bibliographie

[BER 01] BERTELLE C., FLOURET M., JAY V., OLIVIER D., PONTY J.-L., « Automata with multiplicities as behaviour model in multi-agent simulations », *SCI'2001*, 2001.

[BRA 97] BRAZIER F. M. T., DUNIN-KEPLICZ B. M., JENNINGS N. R., TREUR J., « DESIRE : Modelling Multi-Agent Systems in a Compositional Formal Framework », *Int Journal of Cooperative Information Systems*, vol. 6, n° 1, 1997, p. 67–94.

[DIM 92] DIMACS92, « Clique and Coloring Problems, A Brief Introduction, with Project Ideas », 1992, ftp ://dimacs.rutgers.edu/pub/challenge.

[DOR 98] DORNE R., HAO J.-K., « A New Genetic Local Search Algorithm for Graph Coloring », EIBEN A. E., BÄCK T., SCHOENAUER M., SCHWEFEL H.-P., Eds., *Parallel Problem Solving from Nature – PPSN V*, Berlin, 1998, Springer, p. 745–754, http ://citeseer.nj.nec.com/dorne98new.html.

[DRO 93] DROGOUL A., *De la simulation multi-agents à la résolution collective de problèmes : une étude de l'émergence de structure d'organisation dans les systèmes multi-agents*, PhD thesis, Univ. Paris VI, 1993.

[GUT 00] GUTKNECHT O., FERBER J., MICHEL F., « MADKIT : une expérience d'architecture de plateforme multi-agent générique », 2000.

[IGL 99] IGLESIAS C., GARRIJO M., GONZALEZ J., « A Survey of Agent-Oriented Methodologies », MÜLLER J., SINGH M. P., RAO A. S., Eds., *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, vol. 1555, Springer-Verlag : Heidelberg, Germany, 1999, p. 317–330.

[MER 02] MERMET B., « Formal model of a Multiagent System », TRAPPL R., Ed., *Cybernetics and Systems*, Austrian Society for Cybernetics Studies, 2002, p. 653-658.

[VES 00] VESEL A., ZEROVNIK J., « How good can ants color graphs ? », *Journal of computing and Information Technology - CIT*, vol. 8, 2000, p. 131–136, http ://citeseer.nj.nec.com/443529.html.

[WOO 00] WOOLDRIDGE M., JENNINGS N. R., KINNY D., « The Gaia Methodology for Agent-Oriented Analysis and Design », *Autonomous Agents and Multi-Agent Systems*, vol. 3, n° 3, 2000, p. 285-312.