Satisfaction distribuée de contraintes et son application à la génération d'un emploi du temps d'employés*

Thierry Moyaux[1], Brahim Chaib-draa[1], Sophie D'Amours[2]
[1] Centre de recherche sur les technologies de l'organisation réseau (CENTOR), Consortium de recherche FOR@C, Laboratoire DAMAS, Département d'Informatique et de Génie Logiciel, Faculté des Sciences et de Génie, Université Laval, Ste-Foy, Québec, G1K 7P4.
Email : {moyaux, chaib}@iad.ift.ulaval.ca
[2] Centre de recherche sur les technologies de l'organisation réseau (CENTOR), Consortium de recherche FOR@C, Département de Génie Mécanique, Faculté des Sciences et de Génie votre faculté, Université Laval, Ste-Foy, Québec, G1K 7P4.
Email : sophie.damours@gmc.ulaval.ca

Résumé. La génération d'emplois du temps d'employés (ETP pour *Employee Timetabling Problem*) est abordée dans cet article d'un point de vue multiagent afin de donner un cadre d'étude à la coordination distribuée soutenue par la communication. Nous nous sommes penchés plus particulièrement sur le cas des établissements scolaires en cherchant à affecter des professeurs et des salles à des cours en respectant des créneaux horaires. Ce problème d'ETP est formalisé sous la forme d'un problème de satisfaction de contraintes distribué (DCSP pour *Distributed Constraints Satisfaction Problem*). Un algorithme à retour arrière résolvant ce DCSP est alors proposé. Un tel algorithme génère à la fois l'emploi du temps recherché et la coordination des agents qui vont le suivre. Enfin, nous expliquons comment un tel algorithme a été implémenté et validé.

Mots clés : Planification et ordonnancement des opérations, système multi-agent, problème de génération d'emplois du temps, problème de satisfaction distribuée de contraintes.

1. Introduction

Les problèmes de génération de l'emploi du temps des employés (ETP pour *Employee Timetabling Problems*) sont des problèmes très variés. Basiquement, ces problèmes concernent l'assignation de tâches aux membres d'une équipe en tenant compte de contraintes (qualités, contraintes et préférences des employés) et d'objectifs (régulation globale de l'organisation, réduction du coût global, division équitable du travail entre les employés...) (Kragelund et Mayoh, 1999 ; Meisels et Schaerf, 2001). Ces problèmes sont présentés dans la section 2 de ce papier. Depuis près de quarante ans, ce type de problème a attiré l'attention de la communauté scientifique de plusieurs disciplines, dont la recherche opérationnelle et l'intelligence artificielle, et l'intérêt pour ce champ s'est accru lors des dix dernières années (Burke et Petrovic, 2002). Ce problème se modélise généralement par un réseau de contraintes (encore appelé problème de satisfaction de contraintes - en anglais CSP pour *Constraint Satisfaction Problem*) (Kanoh et al., 2001). La résolution d'un CSP consiste à trouver l'assignation consistante de valeurs à des variables prenant leur valeur dans des domaines discrets et finis. Il est à noter que la solution d'un CSP n'est pas forcément la solution optimale (Yung et Yang, 1999). Une fois formalisé par un CSP, un ETP peut être résolu de multiples façons : heuristique spécifique, algorithme génétique, approche multicritère, etc. (Burke et Petrovic, 2002 ; Meisels et Schaerf, 2001 ; Kragelund et Mayoh, 1999 ; Kanoh et al., 2001). Les CSP sont présentés dans la section 3 de ce papier. Alors que les solutions traditionnellement proposées pour résoudre un ETP sont centralisées (c'est la forme classique des CSP), *l'originalité de notre approche est d'être décentralisée*. A cet effet, l'ETP est formalisé sous la forme d'un DCSP (*Distributed Constraint Satisfaction Problem*). Comme le rappelle la section 3, un DCSP est un CSP dans lequel les variables et les contraintes sont distribuées en-

* Nous tenons à remercier le consortium de recherche FOR@C (www.forac.ulaval.ca) pour son soutien financier.

tre de multiples agents (Yokoo et al. 1998). Le problème d'ETP est formalisé sous la forme d'un DCSP, puis un algorithme trouve une solution à ce DCSP. Cet algorithme étant implémenté par un système multi-agent, la solution obtenue correspond à une planification distribuée. En effet, résoudre un DCSP modélisant un ETP revient à coordonner de manière décentralisée un système multi-agent en se basant sur la communication entre les agents. Dans notre approche, le problème d'ETP pour un établissement scolaire est formalisé par un DCSP qui est ensuite résolu par un algorithme à retour arrière. La section 4 présente cette formalisation, cet algorithme, puis le système multiagent que nous avons implémenté à des fins de validation. Enfin, la section 5 contient les résultats de ces tests de validation.

2. Génération automatique d'emplois du temps

La génération d'emplois du temps (ETP pour *Employee Timetabling Problems*) est l'opération d'assignation d'employés à des tâches dans un ensemble d'horaires durant une période fixée, typiquement une semaine. Ce problème est aussi connu sous le nom de problème d'ordonnancement du personnel (SSP pour *Staff Scheduling Problem*) (Kragelund et Mayoh, 1999). Les ETP s'adressent habituellement aux organisations où un ensemble de tâches doivent être accomplies par un ensemble d'employés ayant leurs propres qualifications, contraintes et préférences. L'organisation impose des régulations globales et essaye de réaliser des objectifs eux-aussi globaux tels que la réduction du coût total ou la division équitable de la charge de travail entre les employés (Meisels et Schaerf, 2001). Les recherches dans les ETP ont donné lieu à plusieurs conférences, telles que la série des *Conferences for the Practice and Theory of Automated Timetabling* (PATAT, 2002). Les résultats issus des ETP sont utilisables dans différents secteurs professionnels, tels que l'assignation des horaires à des infirmières, à des policiers, à des pompiers... Chacun de ces secteurs a eu son type de solutions propre. Il semble que les programmes informatiques ne se sont occupés qu'aux plus simples de ces problèmes d'ordonnancement. Cela serait du à la présence fréquente du travail en équipe qui complique grandement le problème. En effet, le travail en équipe implique non seulement un accroissement significatif de la taille de l'espace de recherche à cause de nouveaux types de tâches, mais en plus les contraintes et les objectifs deviennent eux-aussi plus complexes (Kragelund et Mayoh, 1999).

En ce qui nous concerne, nous nous concentrons dans ce document sur les emplois du temps d'établissements scolaires. Burke et ses collègues (Burke et Petrovic, 2002) notent à cet égard que ce type de problèmes se divise en deux catégories principales : les cours et les examens. Différents aspects séparent ces deux catégories. Par exemple, on cherche à regrouper les cours, alors qu'on préfère éloigner les examens les uns des autres le plus possible. Ou encore, un cours peut se tenir à un instant donné dans une salle, alors que plusieurs examens peuvent se tenir en même temps dans la même salle ou un même examen peut être dispatché dans plusieurs salles. Dans la suite, nous ne traitons que des cours. Plus précisément, nous cherchons à affecter des professeurs et des salles à des cours en respectant des créneaux horaires, et cela sur une semaine (et par extension sur une session en recopiant cette semaine autant de fois que nécessaire). C'est-à-dire que chaque cours a besoin exactement d'un créneau horaire et d'une salle, ni plus, ni moins.

3. CSP et DCSP

Les CSP (*Constraint Satisfaction Problem*) sont une classe de problèmes qui ont été formalisés mathématiquement. Une présentation en est donnée dans la prochaine sous-section. Nous nous focalisons ensuite sur les DCSP, qui sont leur pendant distribué et sur lesquels s'appuient notre démarche.

CSP - *Constraint Satisfaction Problem*

Un problème de satisfaction de contraintes (CSP) consiste à trouver l'assignation consistante de valeurs à des variables prenant leur valeur dans des domaines discrets et finis. Il est à noter que la solution d'un CSP n'est pas forcément la solution optimale. Traditionnellement, l'optimisation est faite en sélectionnant la meilleure de ces solutions, et ce après que toutes les solutions possibles aient été propagées (Yung et Yang, 1999).

Différentes recherches (Meisels et Schaerf, 2001 ; Burke et Petrovic, 2002) notent que les contraintes peuvent être découpées en deux : d'une part les contraintes fortes qui doivent absolument être respectées et d'autre part, les contraintes faibles que l'on va essayer de respecter au mieux. Si l'on ne tient compte que des contraintes fortes, le problème consiste donc à trouver une solution (parmi d'autres) qui satisfasse aux contraintes. Si on y ajoute en plus les contraintes faibles, une fonction d'évaluation ou de pénalité mesurant le degré de respect des contraintes faibles est ajoutée. Il s'agit alors de trouver dans l'espace des solutions respectant les contraintes fortes la solution qui maximise (ou, selon les cas, qui minimise) cette fonction. Le problème de satisfaction de contraintes ressemble alors à un problème d'optimisation où l'algorithme cherchera à violer d'abord les contraintes faibles définies comme non-prioritaires, c'est à dire les contraintes faibles ayant un petit poids dans la fonction d'évaluation, avant de violer les contraintes faibles prioritaires. Deux étapes sont donc nécessaires : une détermination de l'espace des solutions respectant les contraintes fortes, puis une recherche dans cet espace de la solution minimisant ou maximisant la fonction d'évaluation. Ce problème ressemble encore plus à un problème d'optimisation lorsque les contraintes fortes et faibles sont mesurées toutes les deux par cette fonction. En effet, la satisfaction de contraintes consiste alors uniquement à optimiser cette fonction : il n'y a plus d'étape de recherche des solutions respectant les contraintes fortes, seule demeure l'étape de minimisation ou de maximisation de la fonction d'évaluation. Par exemple, Kanoh et ses collègues (Kanoh et al., 2001) construisent une fonction de pénalité en donnant un poids très élevé aux contraintes fortes et un poids très faible aux contraintes faibles.

Formellement, un CSP consiste en n variables x_1, x_2, ..., x_n dont les valeurs sont prises respectivement dans des domaines discrets finis D_1, D_2, ... , D_n et en un ensemble de contraintes sur ces valeurs, chaque contrainte étant définie par un prédicat. Plus précisément, la contrainte $p_k(x_{k1}, ... , x_{kj})$ est un prédicat défini sur le produit cartésien $D_{k1} \times ... \times D_{kj}$. Ce prédicat est vrai ssi l'assignation de valeur de ces variables satisfait cette contrainte. Résoudre un CSP revient à trouver une assignation de valeur à toutes les variables, de manière à ce que toutes les contraintes soient satisfaites. Comme de façon générale la satisfaction de contraintes est NP-complète, une exploration par essai et erreur des alternatives est inévitable (Yokoo et Ishida, 1999). Les algorithmes connus qui sont mis en oeuvre lors de la résolution d'un CSP se divisent en deux groupes (Yokoo et Hirayama, 2000) :

1. les algorithmes de consistence, c'est à dire des algorithmes de prétraitement qui améliorent le travail des algorithmes de recherche ;
2. les algorithmes de recherche, parmi lesquels on distingue :
    - les algorithmes à retour arrière (*backtracking algorithms*) ;
    - les algorithmes d'amélioration itérative (*iterative improvement algorithms*) ;

Une autre classification des techniques de résolution d'un CSP est proposée par Yung et Yang (Yung et Yang, 1999). Les trois catégories de cette classification sont : (i) la réduction du problème qui consiste à retirer les valeurs et les étiquettes de valeur redondantes, (ii) la synthèse de solution en construisant incrémentalement un treillis appelé « graphe du problème minimal » qui représente le problème minimal et (iii) la recherche qui inclut les algorithmes à retour arrière (backtracking algorithms), les algorithmes de vérification prévisionnelle (forward checking algorithms), les systèmes de maintenance de la vérité et la propagation de contraintes.

DCSP – *Distributed CSP*

Il existe différentes variantes du CSP original. Par exemple, dans les ICSP (Interval CSP) l'objectif n'est plus de trouver l'assignation des variables qui satisfait aux contraintes, mais les intervalles de ces variables où les contraintes sont respectées (Yung et Yang, 1999). Les systèmes multiagents étant des systèmes distribués, nous nous intéressons plus particulièrement aux travaux effectués dans un autre sous-domaine des CSP : les CSP distribués ou DCSP.

Un DCSP est un CSP dans lequel les variables et les contraintes sont distribuées entre des agents. Ces variables se divisent en deux ensembles disjoints : les contraintes intra-agents et les contraintes inter-agents. Une contrainte intra-agent n'est connue que par un agent. Habituellement, on considère qu'une contrainte inter-agent est connue par tous les agents ayant une variable dans cette contrainte. Comme dans le cas centralisé, résoudre un DCSP consiste à trouver une assignation de valeur aux variables en ne violant aucune contrainte (bien que la littérature des DCSP se concentre principalement à la résolution des contraintes inter-agents). Trouver une assignation de valeur aux variables inter-agents peut être vu comme réaliser la cohérence ou la consistance d'un système multiagent (Bessière et al., 2001 ; Yokoo et al., 1998).

Les heuristiques de résolution d'un DCSP se classent en deux catégories. D'une part, la programmation orientée-marché repose sur les mécanismes d'enchères. D'autre part, des agents peuvent être en compétition pour utiliser des ressources, ces ressources pouvant calculer leur demande aggrégée (Durfee, 1999). En outre, les algorithmes de résolution des CSP semblent similaires à des méthodes de traitement parallèle/distribué pour résoudre des CSP, quoique les motivations de recherche soient fondamentalement différentes (Yokoo et Hirayama, 2000).

4. Présentation de notre approche

Notre approche repose sur un algorithme à retour arrière issu des DCSP pour obtenir de façon distribuée l'emploi du temps recherché. Le DCSP est formulé dans la sous-section 4.1 et l'algorithme dans la sous-section 4.2. Cet algorithme est mis en œuvre dans un système multiagent détaillé dans la sous-section 4.3.

Formulation du DCSP

Nous formalisons ici le problème de l'affectation de professeurs et de salles à des cours en respectant des créneaux horaires prédéfinis. Nous considérons (i) qu'il y a trois créneaux horaires par jour, cinq jours par semaine, (ii) que les professeurs ne sont pas toujours disponibles pour donner des cours, (iii) qu'ils ont un nombre maximum de cours à donner hebdomadairement et (iv) qu'ils ne peuvent pas donner n'importe quel cours. Ce problème de génération se défini par cinq ensembles: *Probleme = {Professeurs, Salles, Horaires, Cours, Contraintes}* où *Professeurs* = $\{P_1, \ldots, P_i, \ldots, P_p\}$ est l'ensemble des p professeurs, *Salles* = $\{S_1, \ldots, S_j, \ldots, S_s\}$ représente l'ensemble des s salles, *Horaires* = $\{H_1, \ldots, H_k, \ldots, H_h\}$ l'ensemble des h créneaux horaires disponibles dans une semaine, *Cours* = $\{C_1, \ldots, C_l, \ldots, C_c\}$ l'ensemble des c cours à donner durant la session considérée et *Contraintes* l'ensemble des contraintes entre les variables des quatre ensembles précédents.

Nous introduisons dans ce paragraphe une notation afin de présenter ensuite l'ensemble *Contraintes* qui est constitué de relations entre les variables des quatre autres ensembles. Si le problème était traité de manière centralisée, on pourrait définir une matrice de binaires notée $X[1..p][1..s][1..h][1..c]$ telle que $\forall P_i \in Professeurs$, $\forall S_j \in Salles$, $\forall H_k \in Horaires$, $\forall C_l \in Cours$, $x[P_i][S_j][H_k][C_l] = 1$ si le professeur P_i est occupé à donner le cours C_l dans la salle S_j pendant l'horaire H_k et $x[P_i][S_j][H_k][C_l] = 0$ dans les autres cas. Comme le problème est traité de façon distribuée, cette matrice X n'existe pas et les informations qui y sont stockées sont dispatchées entre les agents participant à la résolution. Par exemple, le professeur P_i

connaît la matrice $X_{Pi}[1..s][1..h][1..c]$ définie par : $\forall\ S_j \in Salles$, $\forall\ H_k \in Horaires$, $\forall\ C_l \in Cours$, $x_{Pi}[S_j][H_k][C_l] = x[P_i][S_j][H_k][C_l]$ (P_i est fixé). Ce professeur P_i ne connaît donc que le contenu X_{Pi} extrait de X qui le concerne. De la même manière, la salle S_j connaît $x_{Sj}[P_i][H_k][C_l] = x[S_j][P_i][H_k][C_l]$ (S_j est fixé) et le cours C_l connaît $x_{Cl}[P_i][S_j][H_k] = x[P_i][S_j][H_k][C_l]$ (C_l est fixé). La figure 1 illustre le contenu de ces matrices en montrant l'allure de la mémoire X_{Sj} de chaque agent-salle S_j pour un cas à quinze horaires, six cours et deux professeurs où le cours C_2 est affecté au professeur P_1 pendant l'horaire H_{11}. Il est à noter que cette vision est un peu idyllique, car elle sous-entend qu'une même donnée (i.e. une case dans la matrice X) connue par deux résolveurs du DCSP vaut tout le temps la même chose pour chacun de ces deux résolveurs, alors qu'il y a en fait un temps de propagation de l'information (par exemple, un professeur ne sait qu'une salle est occupée qu'après qu'il se soit informé de l'emploi du temps de cette salle). En utilisant ces notations, nous pouvons maintenant définir l'ensemble *Contraintes* de la manière suivante :

- un professeur *Pi* peut donner au plus un cours pendant l'horaire H_k : $\forall\ P_i \in Professeurs$, $\forall\ H_k \in Horaires$, on a $\sum_{Sj \in Salles} (\sum_{Cl \in Cours} x_{Pi}[S_j][H_k][C_l]) \leq 1$.
- un professeur P_i a un maximum *MAX* de cours à donner (mais pas de nombre minimum, ce qui simplifie beaucoup le problème, comme nous le verrons plus loin) : $\forall\ P_i \in Professeurs$, on a $\sum_{Sj \in Salles} (\sum_{Hk \in Horaires} (\sum_{Cl \in Cours} x_{Pi}[S_j][H_k][C_l])) \leq MAX$.
- une salle S_j peut contenir au plus un cours C_l donné par un unique professeur P_i pendant l'horaire H_k, soit : $\forall\ S_j \in Salles$, $x_{Sj}[P_i][H_k][C_l] = 1 \Rightarrow \sum_{Pi' \neq Pi} (\sum_{Hk' \neq Hk} (\sum_{Cl' \neq Cl} x_{Sj}[P_i'][H_k'][C_l'])) = 0$, ce qui est équivalent à : $\forall\ S_j \in Salles$, $\sum_{Pi \in Professeurs} (\sum_{Hk'' \in Horaires} (\sum_{Cl'' \in Cours} x_{Sj}[P_i''][H_k''][C_l''])) \leq 1$.
- un cours C_l peut être donné au plus par un unique professeur P_i dans une unique salle S_j à un unique horaire H_k : $\forall\ C_l \in Cours$, $x_{Cl}[P_i][S_j][H_k] = 1 \Rightarrow \sum_{Pi' \neq Pi} (\sum_{Hk' \neq Hk} (\sum_{Cl' \neq Cl} x_{Cl}[P_i][S_j][H_k])) = 0$.

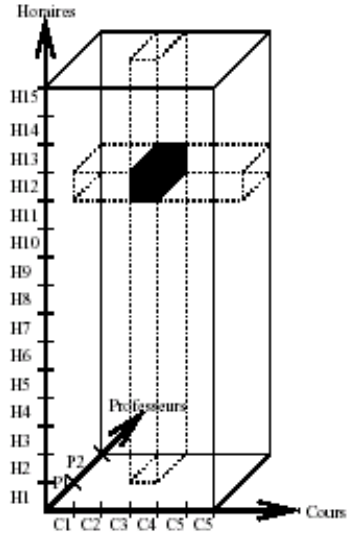

Figure 1. Allure de la matrice X_{sj} de chaque agent-cours S_j.

Algorithme de résolution du DCSP

Il semble que les agents-cours soient les plus à même de résoudre ce DCSP. Dans une première approche, il nous semblait intuitivement que c'étaient aux agents-professeur de faire cette résolution, mais deux problèmes sont alors apparus. En premier lieu, le premier professeur à choisir un cours pourrait « raffler » tous les cours populaires auprès des étudiants. En second lieu, un problème plus sérieux se pose pour s'assurer que chaque cours est bien pris par un professeur. Supposons par exemple que le professeur P_0 soit le seul à être capable d'assurer le cours C_0. Mais ce professeur est le premier à choisir ses affectations, et il ne prend pas ce cours C_0. Les professeurs suivants vont alors prendre tous les autres cours, sauf le cours C_0 puisqu'ils ne sont pas en mesure de le donner. Dans la situation décrite, seuls les agents-professeur sont actifs, ce qui implique que ce sont eux qui devraient se rendre compte collectivement que le cours C_0 a été

oublié et si aucun ne le fait, alors le cours C_0 sera oublié. Au contraire, si comme nous le proposons, ce sont les agents-cours qui résolvent le DCSP, ce sont des professeurs (et non des cours) qui ne seront pas pris, mais ce n'est pas grave vu que le but était de trouver des professeurs pour chaque cours, et nous de trouver des cours pour chaque professeur. Ceci est vrai tant qu'aucune contrainte sur le nombre de cours minimal à donner par chaque professeur n'est ajoutée dans l'ensemble *Contraintes* (cf. sous-section 4.1 page 7).

L'algorithme donné ici est donc implémenté dans les agents-cours. Il s'agit d'une adaptation de l'algorithme d'*Asynchronous Backtracking* (Yokoo et Hirayama, 2000). La principale différence entre ces deux algorithmes semble être dans l'utilisation des priorités : nous ne donnons pas une priorité différente à chaque agent : seuls les agents ne trouvant pas d'affectation compatible avec les contraintes deviennent prioritaires par rapport aux agents qui ont trouvé une affectation. En outre, les agents-salle et -professeur permettent de fournir des informations à cet algorithme : par conséquent, le comportement des agents-salle et des agents-professeur est réactif, car ils ne font que répondre aux questions posées par les agents-cours.

L'idée générale de l'algorithme est illustrée à la figure 2 : pour chaque agent C_l, il s'agit de « noircir » (mettre à la valeur « vrai ») des cases de la matrice X_{Cl} en fonction de la négociation avec les autres agents-cours et des données provenant des agents-professeur et -salle, pour finalement y choisir une case « blanche » (dont la valeur est « faux »). Plus précisément, le mécanisme de la figure 2 se déroule comme suit. Chaque agent-cours C_l commence par récolter les données des agents-professeur et -salle pour mettre à jour sa matrice X_{Cl} (état « S'informer »), puis il cherche dans cette matrice une affectation *{professeur, salle, horaire}* libre (état « Choisir »). S'il n'en trouve pas, il demande à un autre agent-cours qui a lui-aussi trouvé une affectation de renoncer à cette affectation (état « Demander »), puis recommence l'algorithme au début. Si au contraire il trouve une affectation libre, il informe les agents-professeur et -salle concernés de l'horaire qu'il aimerait les retenir (état « Prévenir »). Si l'un de ces deux agents lui répond (ou les deux) qu'ils ne peut pas, l'agent C_l prévient l'autre agent qu'il ne veut plus de cet horaire (état « Annuler ») et recommence l'algorithme au début. Si au contraire ces deux agents sont d'accord, notre agent C_l note que cette affectation est la sienne, puis il informe les autres agents-cours qu'il s'est trouvé

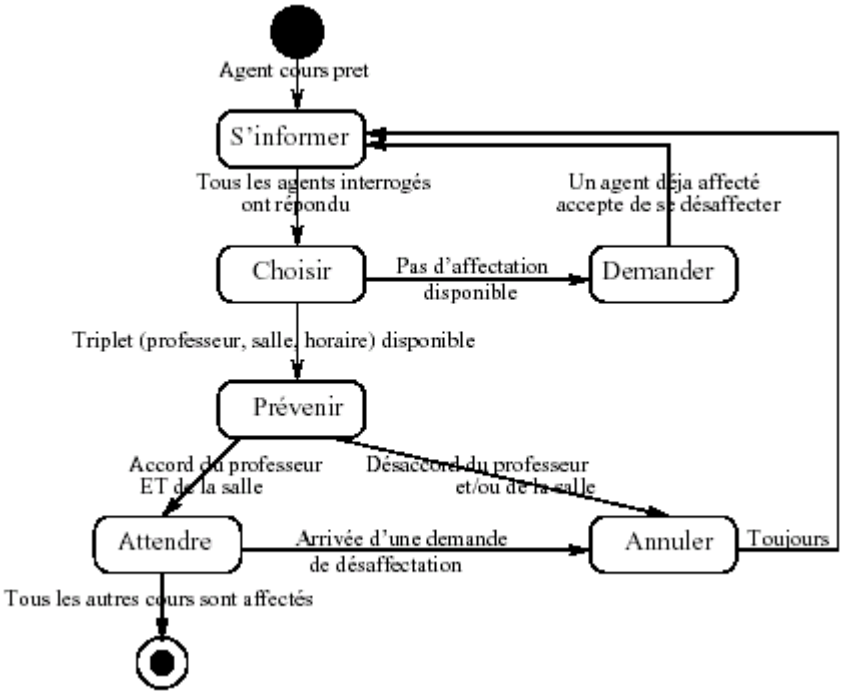

Figure 2. Algorithme à retour arrière.

une affectation et se met enfin en attente d'un message provenant d'un autre agent-cours qui lui demande-

rait de se désaffecter (état « Attendre »). Une fois que tous les agents cours ont averti l'agent C_l qu'ils ont eux-aussi trouvé leur affectation, celui-ci termine son exécution (état final).

Le système multiagent

La résolution du problème de génération automatique d'emplois du temps est implémentée par un système multiagent. Les ressources physiques (professeurs, salles et cours) y sont des agents. Ce sont les agents-cours qui résolvent le problème, alors que les agents-professeur et les agents-salle se contentent de donner aux agents cours les informations qu'ils leur demandent.

La figure 3 présente les caractéristiques du système multiagent. On note tout d'abord que tous les agents ont un environnement uniquement composé des autres agents. Ensuite, les agents-salle et professeur sont très semblables : ils perçoivent les mêmes choses (à savoir les messages venant des agents-cours), n'agissent pas et n'ont pas de but, puisqu'ils ne servent que de drapeau pour bloquer la ressource réelle qu'ils représentent. Les agents-professeur ajoutent seulement la capacité de pouvoir recueillir les préférences des professeurs réels. Enfin, les agents-cours sont les plus évolués. Ils perçoivent les messages en provenance de tous les autres agents du système. Leur principale action est de négocier avec les autres agents-cours, et pour mener à bien cette négociation, leurs actions secondaires sont de s'informer auprès des agents-salle et -professeur. Enfin, leur but est de trouver l'emploi du temps des cours recherché.

Agents	Percepts	Actions	Buts	Environnement
Salle	*Messages venant des agents-cours*	*Aucune*	Aucun	*Tous les autres agents*
Professeur	*1/ Messages venant des agents-cours* *2/ Préférences des professeurs réels*	*Aucune*	*Aucun*	*Tous les autres agents*
Cours	*Tous les autres agents (salles, cours et professeurs) du système*	*1/ Négocie avec les autres agents-cours* *2/ Interroge les agents-salle et -professeur*	*S'affecter un professeur et une salle à un horaire convenable en tenant compte des contraintes*	*Tous les autres agents*

Figure 3. Description du système multiagent.

Élément	*Attribut*	*Valeur*
Agents	*nombre*	*nb. de salles + nb. de cours + nb. de professeurs*
	uniformité	*2 types d'agents (cours vs. professeur et salle)*
	buts	*complémentaires (cours), inexistants (prof. et salle)*
	architecture	*délibérative (cours), réactive (professeur et salle)*
	capacités	*avancées (cours), basiques (professeur et salle)*
Interactions	fréquence	*élevées (cours), basses (professeur et salle)*
	persistence	*long-terme pour tous*
	niveau	*connaissance (cours) et signaux (professeur et salle)*
	motif	*décentralisé*
	objectif	*coopération*
Environnement	*prédictabilité*	*prévisible*
	accessibilité	*illimitée*
	dynamisme	*fixé*
	diversité	*pauvre (2 types d'agents)*
	disponibilité des ress.	*essources contraintes*

Figure 4. Description des agents

La figure 4 présente les caractéristiques des agents en les décomposant en trois catégories. En premier lieu, les agents eux-mêmes sont décrits. On note à ce niveau que, comme sur la figure 3, il y a deux types d'agents. D'un côté, les agents-professeur et -salle ne servent que de drapeau sur une ressource, c'est pour-

quoi ce sont des agents réactifs très simples sans but qui se contentent de dire aux agents-professeurs quand leur ressource est disponible. De l'autre côté, les agents-cours essayent de construire ensemble un emploi du temps en suivant l'algorithme de résolution du DCSP de la figure 2. En deuxième lieu, les interactions qui s'établissent entre les agents sont caractérisées comme suit. La dichotomie précédente entre les deux types d'agents affectent aussi les interactions. En effet, seuls les agents-cours dialoguent de manière soutenue avec tous les types d'agents. En troisième lieu, l'environnement des agents est décrit : l'environnement est prévisible, toutes ses caractéristiques sont accessibles, son dynamisme est constant, il ne contient que des agents et les ressources (accessibles par les agents-professeur et -salle) sont contraintes. Tous les agents-cours sont implémentés de la même manière. Comme ils interrogent les mêmes agents-salles et professeurs, ils disposent de la même information (du moins si on parle de façon générale, puisque, comme nous l'avons déjà mentionné plus haut, il existe un temps de propagation de l'information). Par conséquent, ils vont naturellement avoir tendance à réclamer les mêmes professeurs aux mêmes horaires (ils ne vont pas tous demander exactement les mêmes professeurs à cause de leurs contraintes respectives). Pour limiter le nombre de conflits, et aussi pour répartir uniformément les cours sur toute la semaine, les agents-cours C_l n'essayent pas de s'affecter avec le premier professeur et la première salle qu'ils trouvent dans X_{Cl}, mais ils vont choisir une affectation au hasard parmi les cases marquées libres. Enfin, les communications entre les agents sont limitées. De ce fait, un agent professeur P_i ne peut par exemple pas communiquer sa matrice X_{Pi}. Par conséquent, les données qu'il transmet sont une agrégation du contenu de cette matrice. Cette agrégation se fait selon le destinataire du message. Par exemple, si le destinataire est le cours C_l, alors le professeur P_i ne lui enverra que la ligne d'horaires contenue dans X_{Pi} qui le concerne. Si le professeur P_i ne peut pas donner le cours C_l, alors P_i dira à C_l qu'il n'est jamais disponible, alors qu'en fait il peut très bien être libre pendant l'un ou l'autre horaire, mais pas pour ce cours-là.

5. Expérimentations et résultats

L'algorithme de la figure 2 a été implémenté puis avec une salle S_1, six cours de trois heures notés C_0 à C_5, quinze créneaux horaires de trois heurs notés H_O à H_{15}, deux professeurs notés P_0 et P_1 et les contraintes des tableaux 5 et 6. Le tableau 5 indique par un X les horaires où les professeurs ne peuvent assurer un cours et le tableau 6 donne quels cours peuvent être donnés par chaque professeur et le nombre maximum de ces cours. Le tableau 5 indique par exemple que le professeur P_0 ne travaille pas du tout le lundi, puisque P_0 ne travaille pas aux horaires H_0, H_1 et H_2 (il y a trois créneaux horaires par jour), et le tableau 6 indique que ce même P_0 ne peut pas enseigner les cours C_0, C_1 et C_2. Nous avons surchargé les tableaux 5 et 6 afin d'étudier la convergence de notre algorithme en fonction de l'augmentation des contraintes. Les données de ces deux tableaux correspondent à un cas limite : si l'on ajoute une contrainte, il n'y a plus de solution possible. Les résultats obtenus sont résumés par les deux courbes de la figure 7. La vitesse de convergence qui y est représentée est mesurée en nombre de retours en arrière (*backtracks*), c'est à dire soit en termes de nombre total de refus d'affectation émis par les agents-professeur et -salle, soit en termes de nombre total de demandes de changement d'affectation adressées à un agent-cours par un autre agent-cours.

	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14
P0	X	X	X				X	X	X	X	X	X	X	X	X
P1				X	X	X	X	X	X	X	X	X	X	X	X

Figure 5. Contraintes des agents professeur de notre application.

	C0	C1	C2	C3	C4	C5	Total maxi
P0	X	X	X				3
P1				X	X	X	3

Figure 6. Capacité des professeurs à enseigner un cours.

La « courbe attendue » de la figure 7 illustre les résultats que nous pensions obtenir, à savoir l'accélération progressive et croissante jusqu'à l'infini du temps de convergence de notre algorithme lorsque l'on relâche

les contraintes. Un temps infini signifie que l'algorithme ne trouve jamais de réponse et un temps nul que l'algorithme trouve la solution du premier coup, sans faire de retour en arrière (*backtrack*). La « courbe obtenue » montre que le nombre réel de réaffectations reste toujours très proche de zéro tant que des solutions existent, puis passe brusquement à l'infini quand aucune solution n'est possible. La courbe est une droite à peu près constante sur l'axe des abscisses avant le point $Contraintes_{Max}$.

Concernant la courbe réellement obtenue, nous avons plus précisément constaté un maximum[1] de cinq refus d'affectation définitive émis par les salles ou par les professeurs, et pratiquement jamais de réaffectation demandée à un cours par un autre cours. Dis autrement, une fois que les agents-cours ont choisi un triplet *{horaire, professeur, salle}*, ce triplet est le plus souvent accepté par le professeur et la salle concernés, et aucun autre cours ne remet ce choix en cause par la suite. Notre algorithme converge donc vite sur le cas étudié[2], alors que les *backtracks* n'ont pas été optimisés (un exemple d'optimisation pourrait être qu'un agent-cours C_0 qui demande à un agent-cours C_1 de se désaffecter pourrait choisir C_1 parce qu'il occupe une salle ou un professeur intéressant C_0). En particulier, s'il n'existe qu'une unique solution dans un espace de recherche très grand, cette absence d'optimisation pourrait mener l'algorithme à tourner en rond, car rien ne le guide vers la solution. Cela nous mènerait à l'obtention de la « courbe attendue » de la figure 7 et nous pensons que c'est plus particulièrement le nombre de cours qui doit être augmenté pour obtenir cette courbe; si c'est le cas, cela pourrait fixer des limites à l'extensibilité de notre algorithme en interdisant la résolution de problèmes dépassant une certaine taille.

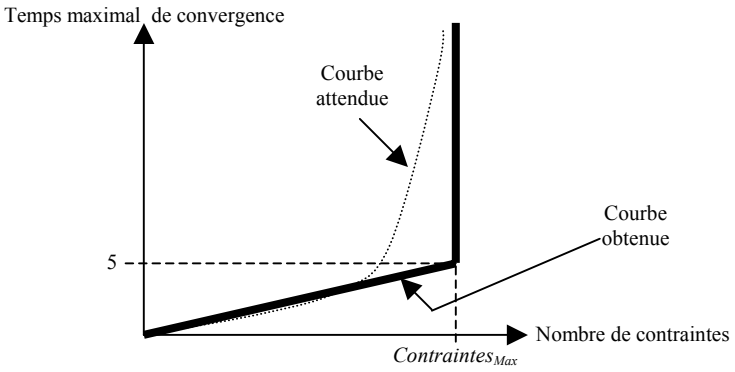

Figure 7. Vitesse de convergence de notre algorithme distribué.

De plus, rien dans notre algorithme n'est prévu pour détecter les cas où il n'y a pas de solution. Par exemple, le tableau de contraintes 6 doit contenir au moins une case sans X par colonne pour qu'une solution existe, car il faut au moins un professeur capable d'enseigner chaque cours pour que le problème de départ soit soluble. Si ce n'est pas le cas, la génération d'un emploi du temps n'est pas possible. Une telle absence de solution se traduit dans notre programme par des agents-cours qui se demandent indéfiniment les uns aux autres de se désaffecter mutuellement « pour se faire de la place ». Le système multiagent ne s'arrête alors jamais.

En outre, notre algorithme est complet, c'est à dire qu'il ne perd pas de solution lors de son exécution. En effet, une fois qu'un cours s'est choisi une affectation *{horaire, professeur, salle}* convenable, il l'abandonne complètement aussitôt qu'un autre cours qui ne se trouve pas d'affectation le lui demande. Lors de ce changement, cet agent-cours reprend ses recherches à zéro. Il serait intéressant de prouver comme dans (Bessière et al., 2001). Enfin, l'espace mémoire nécessaire à notre algorithme est de l'ordre de $p \times s \times h$ par agent-cours (puisque cet algorithme est implémenté dans un agent-cours). Pour des cas pratiques, cet espace nécessaire est négligeable devant la taille des agents, car occupé par des booléens.

---

[1]Nous donnons le maximum obtenu, entendu que ce chiffre vaut la plupart du temps zéro et qu'il peut varier d'une expérience à l'autre du fait que l'ordre d'exécution des agents et les affectations qu'ils choisissent peuvent différer.

[2] Nous ne sommes pas parvenus à réduire significativement la vitesse de convergence en modifiant le cas étudié.

6. Conclusion

Ce papier a présenté un problème de génération distribuée de l'emploi du temps des cours d'un établissement scolaire. Résoudre un tel problème revient à coordonner des agents de façon distribuée en se basant sur la communication. A cet effet, le problème de départ a été formalisé sous la forme d'un problème de satisfaction de contraintes distribué (DCSP pour *Distributed Constraints Satisfaction Problem*), puis un algorithme à retour arrière a été proposé pour le résoudre et enfin cet algorithme a été implémenté et testé dans un système multi-agent.

7. Bibliographie

Bessière, C., A. Maestre and P. Meseguer (2001). Dynamic backtracking distribué. In : *JNPC'01* (K. Sjöström and L.-O. Rask, ed.), pp. 61-72, Toulouse, France.

Burke, E. K. and S. Petrovic (Accepted for publication in 2002). Recent research directions in automated timetabling, *European Journal of Operational Research*.

Durfee, E. H.(1999). Distributed problem solving and planning. In: *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*, pp. 121-164, The MIT Press, third edition.

Kanoh, H., M. Kondo and M.Sugimoto (2001). Solving timetabling problems using genetic algorithms based on minimizing conflict heuristics. In: *EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, Athens, Greece.

Kragelund, L. and B. Mayoh (1999) Nurse scheduling generalised. submitted *Art. Int. Medicine*.

Meisels, A. and A. Schaerf (accepted 2001) Modelling and solving employee timetabling problems. *Appllied Intelligence*.

PATAT (2002). *International Series of Conferences on the Practice And Theory of Automated Timetabling*. http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml.

Yokoo, M., E. H. Durfee and T. Ishida (1998) The distributed constraint satisfaction problem: Formalization and Algorithms. *IEEE Trans. on Knowledge and Data Eng.*, Vol. 10, pp. 673-385.

Yokoo, M. and T. Ishida (1999) Search Algorithms for Agents. In: *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence,* pp. 165-199, The MIT Press, third edition.

Yokoo, M. and K. Hirayama (2000) Algorithms for distributed constraint satisfaction: A review. In: *Autonomous Agents and Multi-Agent Systems*, Vol. 3.

Yung, S. K. and C. C. Yang (1999) A new approach to solve supply chain management problem by integrating multi-agent technology and constraint network. In: *Proc. 32nd Hawaii International Conference on System Sciences*.