Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

Résolution d'emploi du temps dynamique et distribuée par auto-organisation coopérative

Gauthier Picard

IRIT - Université Paul Sabatier - Toulouse III
118, route de Narbonne
31062 Toulouse Cedex, France
picard@irit.fr

Résumé : La coopération est un moyen pour les systèmes multi-agents de fonctionner plus efficacement et de manière plus adaptative. Elle peut être vue comme un critère local de réorganisation pour les agents afin de produire une fonction globale et collective plus adaptée. Cet article montre une application des comportements coopératifs à un problème de résolution d'emploi du temps, ETTO, dans lequel la satisfaction de contrainte est distribuée dans des agents coopératifs. Cette application a été prototypée et montre des résultats positifs en terme d'adaptation, de robustesse et d'efficacité de cette approche.

Mots-clés : Systèmes multi-agents, auto-organisation, coopération

1 Introduction

En réponse à la complexité grandissante des environnements logiciels –en nombre de participants ou en dynamique– les systèmes artificiels sont de plus en plus difficiles à concevoir convenablement. La fonction globale de ces systèmes est souvent spécifiée incomplètement ou de manière floue, mais les parties restent facilement identifiables et les théories de micro-niveau les caractérisant sont connues. Des tels phénomènes, qualifiés d'*émergents*, sont étudiés par les biologistes et les physiciens depuis des années. Les deux principales propriétés de ces systèmes sont l'*irréductibilité* des macro-théories aux micro-théories (Ali *et al.*, 1997) et les *mécanismes auto-organisateurs* qui sont à l'origine de l'adaptation et de l'apparition de nouvelles propriétés émergentes (Goldstein, 1999).

Les cartes de Kohonen ou les algorithmes fourmis sont deux exemple pertinents de transcriptions de mécanismes auto-organisateurs (Kohonen, 2001; Bonabeau *et al.*, 1997). Pour être appliqués à des tâches moins spécifiques, les mécanismes nécessitent de fournir aux parties des capacités cognitives afin de décider quand se réorganiser pour s'adapter à la pression de l'environnement et atteindre le but global. Les parties deviennent alors des agents. En réponse à ce besoin de prise de décision, l'approche par *systèmes multi-agents adaptatifs* (ou *AMAS*) propose la *coopération* comme critère local –pour les parties ou agents– de réorganisation. Ici, la coopération n'est pas

Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

limitée au partage de ressources ou de tâches, mais est un directive comportementale. De plus, la coopération est vue de manière proscriptive : les agents doivent localement changer leurs interactions lorsqu'il sont en *situation non coopérative* (ou *NCS*). Ces changements locaux peuvent être vus, au niveau global, comme une réorganisation du système. Dans un AMAS, un agent est coopératif s'il vérifie les trois méta-règles suivantes (Camps *et al.*, 1999) :

c_{per} : les signaux perçus sont compris sans ambiguïté ;
c_{dec} : les informations reçues produisent un raisonnement ;
c_{act} : le raisonnement produit une action utile à autrui.

Si un agent détecte qu'il est en NCS ($\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$), il doit agir pour revenir à un état coopératif. Le théorème de l'adéquation fonctionnelle (Georgé *et al.*, 2004) assure que le système produit une fonction globale correcte –avec aucune interaction inutile ou antinomique avec son environnement– si tous les agents sont coopératifs.

Par conséquent, concevoir des systèmes adaptatifs revient à fournir des agents coopératifs et ainsi assurer l'adéquation fonctionnelle du système. Dans les sections suivantes, cette approche est illustrée en définissant des comportements coopératifs pour des agents devant résoudre dynamiquement un problème d'emploi du temps universitaire. Les enseignants et les groupes d'étudiants doivent trouver des partenaires, des créneaux horaires et des salles pour donner ou recevoir des enseignements. Chaque acteur possède des contraintes concernant ces disponibilités ou des équipements nécessaires. De plus, un enseignant peut ajouter ou retirer des contraintes à n'importe quel moment de la résolution via une interface adaptée. Un telle application nécessite clairement de l'adaptation et de la robustesse. Le système doit être capable de s'adapter aux perturbations environnementales (modifications de contraintes) et ne pas calculer de nouvelles solutions, depuis le début, à chaque changement. L'organisation collective adéquate doit émerger des interactions locales entre acteurs.

Le système de résolution a été nommé ETTO, pour *Emergent TimeTabling Organization*. Deux types d'agents ont été identifiés et sont présentés dans la section 2. Ces agents respectent plusieurs règles de coopération qui sont exposées dans la section 3. Des expérimentations montrent des résultats sur l'adaptation et la robustesse de cette approche dans la section 4.

2 Les agents dans ETTO

Deux différentes classes d'agents ont été identifiées dans ETTO : les *Representative Agents* (RA) et les *Booking Agents* (BA). Les RAs délèguent l'exploration de l'espace des solutions (une grille à n dimension de cellules représentant le planning) aux BAs. Chaque cellule c_i de la grille est contrainte (créneau, nombre de places, etc), ce qui est regroupé dans un ensemble $C(c_i)$. La coopération entre les agents doit mener à une organisation correcte en explorant efficacement la grille.

2.1 Representative Agents

Les RAs forment l'interface entre les acteurs humains (enseignants ou étudiants) et le système de résolution d'emploi du temps. Ils possèdent des contraintes (qualifiées

Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

d'*intrinsèques*) concernant la disponibilité, les nécessités d'équipements (projecteurs, tableaux blancs, etc) ou tout autre type de contrainte personnelle. Pour explorer efficacement les possibilités de partenariat et de réservation de salles, ces agents délèguent l'exploration aux BAs. Un RA (appelé *proxy*) créé autant de BAs (appelés *délégués*) qu'il a de cours à donner ou à recevoir. Il place initialement ces BAs de manière aléatoire dans la grille du planning. Les BAs d'un même RA sont appelés *frères*. La cohérence entre des délégués frères est assurée par leur proxy.

La tâche d'un RA est simple : prévenir ses BAs délégués lorsque l'utilisateur ajoute ou supprime des contraintes et d'informer tous ses BAs délégués lorsqu'un de ses BAs délégués produit une nouvelle contrainte (dite *induite*) suite à une réservation, par exemple, ce qui doit inciter ses frères à ne pas réserver au même créneau horaire.

2.2 Booking Agents

Les BAs sont les véritables acteurs de l'auto-organisation dans ETTO. Ils doivent réserver les créneaux et les salles et trouver des partenaires (étudiants pour les enseignants et *vice versa*) en accord avec les contraintes de leur proxy.

En situation coopérative, un BA, qui est dans une cellule de la grille (i.e. un créneau horaire dans une salle pour un jour donné) la réserve et établit un partenariat avec un autre BA. Mais cette situation nominale n'est pas assurée au commencement car les BAs sont positionnés aléatoirement. Les BAs ont alors besoin de se réorganiser dans la grille en changeant leurs partenariats et réservations, pour atteindre un emploi du temps qui leur paraît adéquat. De telles situations sont des NCS. Par conséquent, les BAs doivent être capables de répondre à ces situations en respectant des règles de coopération (voir section 3).

Les actions qu'un BA peut effectuer sont simples : établir (ou annuler) un partenariat avec un autre BA, réserver (ou libérer) une cellule (en marquant ou supprimant un marqueur avec son adresse), se déplacer dans une autre cellule et envoyer des messages aux autres agents qu'il *connaît*. Un BA ne connaît que son proxy et les BAs qu'il rencontre dans les cellules, en cours de résolution.

Le cycle de vie d'un BA est un cycle classique "perception-décision-action" comme proposé dans (Capera *et al.*, 2003) :

1. Durant la phase de *perception*, le BA vérifie les messages reçus (des autres BAs ou de son proxy) et met à jour les données concernant la cellule qu'il occupe (les BAs dans la cellule, les marqueurs, les propriétés de la cellule) ;
2. Durant la phase de *décision*, le BA doit choisir une action à exécuter pour être le plus coopératif possible, en accord avec les règles de coopération ;
3. Durant la phase d'*action*, le BA exécute l'action choisie.

Pour effectuer ses tâches, un BA possède les propriétés, capacités et connaissances *locales* suivantes :

- sa position courante dans la grille ($cell(ba_i)$) ;
- son partenaire courant ($partnership(ba_i, ba_j)$ avec $i \neq j$) ;
- sa réservation courante ($reservation(ba_i, c_j)$) ;
- son proxy ($proxy(ba_i)$) ;

Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

- son temps de recherche ($time(ba_i)$) pour une réservation ;
- le créneau horaire correspondant à une cellule ($slot(c)$) ;
- une mémoire limitée de BAs connus ($knows(ba_i, ba_j)$) pour leur envoyer des messages, qui est vide au début de la résolution et qui se met à jour au cours de l'exploration de la grille ;
- un ensemble de contraintes intrinsèques (CI_{ba_i}) hérité du proxy à la création du BA ;
- un ensemble de contraintes induites par ses frères (CB_{ba_i}) qui sont attachées et mises à jours par le proxy lorsqu'un de ses frères réserve une cellule pour éviter les situations d'ubiquité (deux BAs d'un même RA réservent deux cellules correspondant à un même créneau horaire, par exemple) ;
- un ensemble de contraintes induites par son partenaire (CP_{ba_i}) qui sont créées et mises à jour à chaque partenariat pour prendre en compte les préférences du partenaire ;
- un ensemble de contraintes induites par sa réservation (CR_{ba_i}) pour éviter des partenariats avec des BAs non disponibles pour la réservation ;
- un ensemble de contraintes d'un premier ensemble qui sont non compatibles avec les contraintes d'un second ensemble ($nonCompatible(C_i, C_j) \subseteq C_i$) ;
- une fonction de pondération des contraintes ($w(c_i) > 0$). Plus le poids est élevé, plus la contrainte est difficile à relâcher. En conséquence, une contrainte est non relâchable si $w(c_i) = +\infty$.

Nous définissons la macro NC afin de simplifier les notations futures :

Définition 1
L'ensemble des contraintes non compatibles entre deux BAs est $NC_{ba_i,ba_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CR_{ba_i}, CI_{ba_j} \cup CB_{ba_j} \cup CR_{ba_j})$.

Pour déterminer les contraintes non compatibles entre deux BAs, les contraintes provenant des partenaires (CP) ne sont pas prises en compte. De la même manière, pour déterminer si une cellule est compatible avec les contraintes d'un BA, les contraintes de la réservation courante du BA ne sont pas incluses :

Définition 2
L'ensemble des contraintes non compatibles entre un BA et une cellule est $NC_{ba_i,c_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CP_{ba_i}, C_{c_j})$.

En utilisant NC, deux porteurs de contraintes (BA ou cellule) peuvent savoir s'ils sont compatibles :

Définition 3
$compatible(x, y) \equiv (NC_{x,y} = \emptyset)$.

Avant de commencer la résolution, il n'y a pas de moyen absolu de décider quels sont les sous-problèmes les plus difficiles à résoudre. De plus, le degré de difficulté peut évoluer à cause de la dynamique induite par l'environnement. Par conséquent, durant le processus de résolution, chaque agent doit être capable d'évaluer la difficulté qu'il a à trouver un partenaire ou une réservation. Un BA ba_i peut calculer le coût d'une

réservation d'une cellule c_j ($rCost(ba_i, c_j)$) et le coût d'un partenariat avec un autre BA ba_j ($pCost(ba_i, ba_j)$) comme suit :

- $rCost(ba_i, c_j) = (\sum_{c \in NC_{ba_i, c_j}} w(c))/time(ba_i)$;
- $pCost(ba_i, ba_j) = \sum_{c \in NC_{ba_i, ba_j}} w(c)$;

Diviser par le temps de recherche, $time(ba_i)$, permet de définir le BA le plus prioritaire : celui qui cherche une cellule depuis le plus longtemps. En effet, informellement, il semble plus coopératif de privilégier les agents ayant du mal à trouver une place dans l'organisation.

2.3 Comportement basique

Les BAs ont deux buts orthogonaux : *trouver un partenaire* et *trouver une réservation*. L'algorithme principal de résolution est distribué dans les BAs et repose sur la coopération entre les agents. La résolution est le résultat des interactions dynamiques entre les entités distribuées (BAs). Comme les BAs doivent atteindre deux buts individuels, le comportement nominal peut être exprimé en termes d'atteinte de ces buts :

Algorithme 1 – Comportement nominal pour un BA.

```
while alive do
  processMessages()
  if partner AND reservation then
    if reservation is optimal then
      moveTo(reservedCell)
    else
      processCurrentCell()
    endif
  else
    moveTo(nextCell);
    addBAsToMemory();
    processEncounteredBAs();
    if NOT (reservation OR partner) then
      processCurrentCell()
    endif
  endif
done
```

Durant la phase de perception, le BA vérifie sa boîte aux lettres, dans laquelle les autres BAs déposent des messages à propos de partenariats ou de réservations. Si le BA a atteint son but (partenariat et réservation), en conséquence des messages reçus, il se déplace jusqu'à sa cellule réservée seulement si sa réservation n'est pas contrainte. Dans le cas où il a relâché des contraintes, il continuera à explorer la grille pour trouver une meilleure solution. Si le BA manque de partenaire ou de réservation, il explore la grille et analyse les BAs rencontrés en mémoire et les cellules connues, i.e. il vérifie si les BAs rencontrés ou les cellules visitées pourraient satisfaire ses propres contraintes.

Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

2.4 Gestion des contraintes et travaux relatifs

Les actions peuvent mener à ajouter des contraintes induites. Par exemple, un BA qui réserve une cellule correspondant à un créneau horaire pour un jour donné avertit ses frères, via son proxy, que ce créneau horaire est inaccessible pour éviter les situations d'ubiquité. A contrario, si un BA annule une réservation, il doit en informer ses frères. Ainsi, un BA doit manipuler deux types de contraintes : les intrinsèques, provenant de l'acteur représenté par son RA proxy, et les induites, provenant de ses frères, de son partenaire et de sa réservation. Bien sûr, certains problèmes n'ont pas de solution sans relaxation de contraintes. En conséquence, les BAs doivent être capables d'affecter des priorités aux contraintes, comme dans les CSPs pondérés ou flous (Bistarelli *et al.*, 1999). Mais contrairement aux CSPs dynamiques classiques (Dechter *et al.*, 1991), la mémoire des états précédents est distribuée parmi les BAs. Enfin, contrairement à toutes ces approches, les BAs raisonnent uniquement sur un nombre limité de BAs connus pour trouver une bonne solution comme dans les CPSs distribués (Yokoo *et al.*, 1998). Comme les BAs sont des agents, ils n'ont pas de connaissance globale. Ainsi, la satisfaction de contraintes est partagée par les BAs, et la solution émerge de leurs interactions point-à-point locales. Cependant, notre approche reste différente des approches susmentionnées, car l'objectif principal n'est pas de fournir un algorithme complet et correct, mais de proposer des mécanismes locaux et robustes, capables de mettre en oeuvre une résolution globale. De manière similaire aux approches par algorithmes fourmis pour résoudre les problèmes d'emploi du temps (Socha *et al.*, 2002), les BAs altèrent leur environnement (la grille) grâce à des marqueurs pour indiquer la cellule réservée et pour contraindre les autres agents. La principale différence avec l'usage de phéromones est la façon dont les marqueurs disparaissent. Chez les fourmis artificielles, les marqueurs (phéromones) s'évaporent avec le temps avec une vitesse difficile à déterminer pour le concepteur. Dans notre algorithme, les marqueurs sont supprimés par des critères déterminismtes à la suite d'une négociation entre des BAs lors d'un conflit de réservation (voir section 3.4), par exemple.

3 Règles d'auto-organisation coopérative

L'algorithme de résolution distribuée de CSP que nous proposons est distribué dans les BAs et repose sur la coopération. Comme précédemment dit dans la section 2, le comportement nominal d'un agent n'est pas suffisant pour mener le collectif vers une organisation adéquate. Les BAs doivent respecter des règles de coopération pour atteindre un état global correct[1]. Concevoir des agents coopératifs revient à implémenter les méta-règles de coopération (voir section 1). Cinq situations pour la réorganisation sont identifiées. Les deux premières sont des situations qui ne respectent pas la méta-règle c_{dec}. Les trois suivantes ne respectent pas c_{dec} Dans l'exemple d'ETTO, comme nous ne nous focalisons que sur les BAs, ils n'y a pas de violation de la règle c_{per} car tous les agents sont identiques et peuvent se comprendre.

L'idée est de concevoir ces règles comme des *exceptions* en programmation objet

[1] i.e. le système produit une fonction sans composante antinomique ou inutile pour son environnement.

classique, au niveau des agents et non au niveau des instructions. Ce concept convient parfaitement à l'approche proscriptive proposée dans (Capera *et al.*, 2003). Comme pour les exceptions, les concepteurs doivent spécifier la condition de déclenchement et l'action à effectuer en retour. Les règles suivantes de coopération sont ainsi présentées comme des paires condition-action. Les conditions ne sont pas forcément exclusives. Cependant, il faut prendre soin de définir une politique de choix des actions en cas de SNC multiples. Nous considérons dans la suite du papier, que la priorité va à c_{dec} puis à c_{act}.

3.1 Incompétence de partenariat

Un des buts à atteindre pour un BA est de trouver un partenaire. Si un BA ba_i rencontre, dans une cellule, un autre BA ba_j avec lequel il ne peut signer de partenariat, ba_i est *incompétent*, en utilisant la terminologie AMAS (Capera *et al.*, 2003). Par exemple, un BA représentant un cours d'un enseignant rencontre un autre BA représentant un cours d'un autre enseignant. Cette NCS d'incompétence de partenariat doit conduire vers une réorganisation collective. Comme la seule entité capable de détecter cette situation est l'agent lui-même, ce dernier est le seul à pouvoir changer l'état de l'organisation. Dans ce cas, l'agent a besoin de changer sa position pour rencontrer d'autres partenaires potentiels. De plus, pour permettre une exploration plus efficace des possibilités de partenariat, ba_i mémorise l'adresse et les BAs connus de ba_j pour les échanger au cours de futures rencontres. Cette règle d'auto-organisation coopérative s'exprime comme suit :

Nom : **Incompétence de partenariat** (pour ba_i)
Condition : $\exists j(j \neq i \wedge knows(ba_i, ba_j) \wedge (\neg compatible(ba_i, ba_j) \vee (pCost(ba_i, ba_j) \geq pCost(ba_i, partnership(ba_i)))))$
Actions : `memorize(ba`$_i$`,knows(ba`$_j$`)); move`

La comparaison des coûts de partenariats ($pCost$) permet à ba_i de décider si le nouveau partenariat potentiel avec ba_j est moins contraignant que le partenariat courant.

3.2 Incompétence de réservation

Similairement à l'incompétence de partenariat, les BAs doivent être capables de changer l'organisation lorsque leur réservation n'est pas pertinente. Cette NCS d'incompétence arrive quand un BA ba_i occupe une cellule dont les contraintes ne sont pas compatibles avec les siennes. Par exemple, un BA représentant un cours d'un enseignant est dans une cellule ($cell(ba_i)$) représentant une salle n'ayant pas assez de places pour accueillir le cours. Dans ce cas, ba_i doit se déplacer dans la grille pour explorer l'espace des possibilités de réservations :

Nom : **Incompétence de réservation** (pour ba_i)

Condition :
$\neg compatible(ba_i, cell(ba_i)) \vee (rCost(ba_i, cell(ba_i)) \geq rCost(ba_i, reservation(ba_i)))$

Actions : `memorize(ba`$_i$`,cell(ba`$_i$`)); move`

Pour améliorer l'exploration de la grille, les BA mémorisent les cellules qu'ils ont parcourues dans lesquelles une NCS est survenue pour les partager lors de rencontres futures et afin de les éviter lors de leur exploration à venir.

3.3 Conflit de partenariat

Les situations dans lesquelles un BA désire signer un partenariat avec un autre BA qui a déjà un partenaire peuvent arriver. Ces situations sont des conflits de partenariat qui violent la condition c_{act} des AMAS. Par conséquent les agents doivent être capable de les détecter et de les résoudre :

Nom : **Conflit de partenariat** (pour ba_i)

Condition :
$\exists j \exists k (j \neq i \wedge i \neq k \wedge knows(ba_i, ba_j) \wedge compatible(ba_i, ba_j) \wedge partnership(ba_j) = ba_k)$

Actions :

```
if (pCost(ba_i,ba_j) < pCost(ba_i,partnership(ba_i)))
   then partner(ba_i,ba_j)
   else move
```

Dans ce cas, la coopération est directement incluse dans l'action de résolution : le partenariat sera signé avec l'agent qui a le plus de difficultés à trouver des partenaires (en comparant les $pCost$).

Lorsqu'il signe un partenariat, un BA informe son partenaire précédent et son proxy. Comme cet algorithme est distribué, l'action `partner(ba`$_i$`,ba`$_j$`)` doit être atomique (au sens de l'accès à une section critique), mais est composé des instructions suivantes :

```
unpartner(ba_i,partnership(ba_i));
setPartner(ba_i,ba_j);
inform(partnership(ba_i));
inform(proxy(ba_i))
```

3.4 Conflit de réservation

Comme pour le partenariat, la réservation peut conduire à un conflit : un BA désire réserver une cellule déjà réservée. Un conflit de réservation peut être spécifié comme suit :

Nom : **Conflit de réservation** (pour ba_i)

Condition :
$\exists j(j \neq i \wedge (reservation(ba_j, cell(ba_i)) \vee \exists k(reservation(ba_j, c_k) \wedge slot(cell(ba_i)) = slot(c_k) \wedge proxy(ba_i) = proxy(ba_j))) \wedge compatible(ba_i, cell(ba_i))$

Actions :

```
if (rCost(ba_i,cell(ba_i)) < rCost(ba_i,reservation(ba_i)))
   then book(ba_i,cell(ba_i))
   else move
```

Lorsqu'il réserve une cellule, un BA doit avertir son partenaire et son proxy afin de les informer de ne pas réserver au même créneau horaire. Comme pour l'action `partner(ba_i,ba_j)`, l'action atomique `book(ba_i,cell(ba_i))` est composée des instructions suivantes :

```
unbook(ba_i,reservation(ba_i));
setBook(ba_i,cell(ba_i));
inform(partnership(ba_i));
inform(proxy(ba_i))
```

3.5 Inutilité de réservation

Dans le cas où un BA est dans la même cellule que l'un de ses frères (même proxy), réserver est inutile. Par conséquent, il peut quitter la cellule sans l'analyser :

Nom : **Inutilité de réservation** (pour ba_i)

Condition : $cell(ba_i) = cell(partnership(ba_i))$

Actions : `processEncounteredBAs(); move`

Analyser les BAs rencontrés (`processEncounteredBAs()`) correspond à l'analyse de la mémoire (liste) des BAs précédemment rencontrés ou partagés par d'autres agents.

4 Prototypage et expérimentations

Pour valider l'algorithme distribué que nous proposons, un prototype d'ETTO a été développé et plusieurs tests ont été effectués pour souligner l'influence de la cardinalité (le nombre d'agents), le bénéfice de la résolution dynamique et la robustesse. Les expérimentations sont basées sur un cahier des charges pour l'emploi du temps établi par le groupe de travail ASA de l'AFIA[2]. Ce cahier des charges est décomposé en quatre variantes : du simple problème sans relâchement de contrainte aux systèmes ouverts par ajout et suppression d'agents en cours de résolution. Pour chacune d'entre-elles, nous obtenons une solution – non unique dans bien des cas.

[2] `http ://www-poleia.lip6.fr/~guessoum/asa/BenchEmploi.pdf`

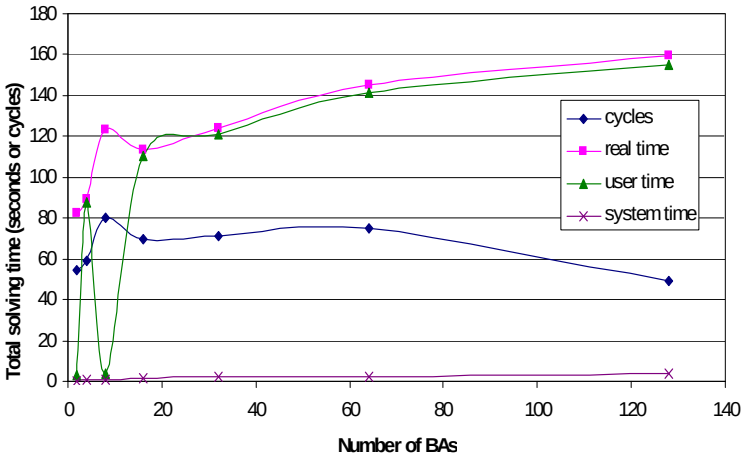

FIG. 1 – Variation du temps de résolution en fonction du nombre d'agents BAs.

4.1 Influence de la cardinalité

La coopération est un liant collectif pour améliorer les interactions entre agents. Ainsi, elle devient un critère pertinent de réorganisation dans des systèmes ayant une cardinalité minimum. La figure 1 montre l'évolution du temps de résolution en fonction du nombre d'agents dans le système. Pour ces expérimentations, nous gardons la même taille d'espace de recherche en augmentant le nombre de cellules de la grille en fonction du nombre d'agents. Seules des contraintes de disponibilités sont attachées aux enseignants : un créneau horaire par jour est interdit.

Une fois le maximum atteint (à 8 BAs), le nombre de cycles (durant lesquels chaque agent agit une fois) décroît lorsque le nombre d'agents augmente. Comme l'espace garde la même dimension, les agents trouvent facilement des partenaires. La mesure de temps qui varie le moins est le temps réel d'exécution. Par conséquent, c'est l'indicateur le plus pertinent de l'évolution du temps de résolution. Au-delà de 32 agents, il suit une courbe logarithmique. Plus il y a de BAs dans le système, plus la résolution est efficace – si une solution existe.

4.2 Relâchement de contraintes

Dans ces expérimentations, les agents doivent relâcher des contraintes pour trouver une solution. La figure 2 montre l'efficacité de la résolution d'ETTO pour une variante à 36 agents (BAs) nécessitant un relâchement de contraintes. Les réservations sont établies après les partenariats. ETTO trouve une solution avec un coût de contraintes relâchées de 10 en 265 cycles. Ce coût représente la somme des poids de toutes les contraintes relâchées par les agents. Néanmoins, le prototype actuel ne gère pas le partage coopératif de cellules lors des négociations et par conséquent lorsqu'un BA se déplace il le fait de manière aléatoire.

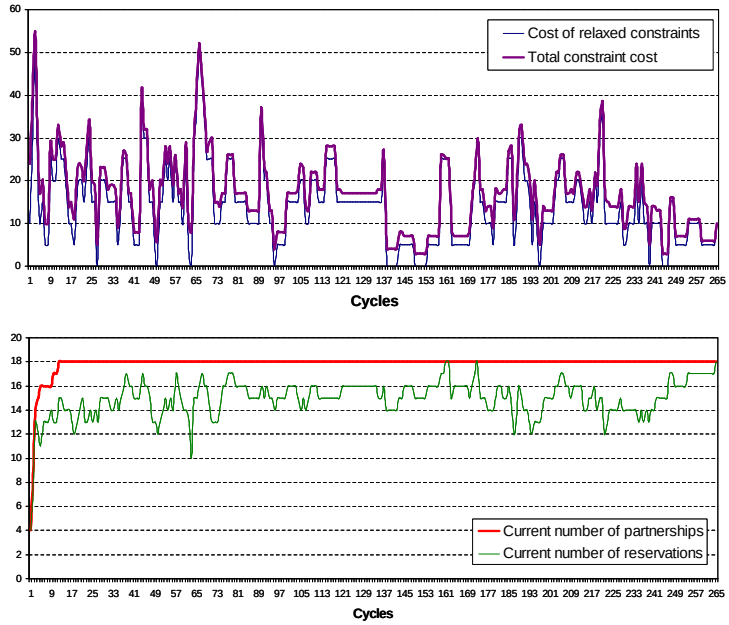

FIG. 2 – Variation du coût global (*en haut*) et des partenariats (*en bas*) durant la résolution d'un problème nécessitant une relaxation de contraintes.

4.3 Résolution dynamique

Les deux premières expérimentations montrent l'apport de l'utilisation de la coopération pour obtenir une résolution efficace de l'emploi du temps. Nous avons effectué une troisième expérimentation pour tester la robustesse à la dynamique de l'environnement. Ici les contraintes apparaissent et disparaissent en cours de fonctionnement. Les contraintes de disponibilités des acteurs ou des salles peuvent aussi évoluer. De plus, des agents peuvent apparaître ou disparaître. En prenant en compte la modélisation choisie, ajouter des contraintes n'est pas différent d'ajouter des agents qui portent des contraintes.

La figure 3 montre des résultats d'une expérimentation avec 36 BAs initiaux. Au cycle 364, à la stabilisation du système, 8 BAs sont retirés de la grille, augmentant ainsi le coût des contraintes relâchées. 20 cycles plus tard, 8 nouveaux agents sont placés aléatoirement dans la grille avec des contraintes adéquates. Le système effectue seulement 7 cycles (du 384 au 391) pour retrouver une organisation adéquate avec un coût nul.

4.4 Discussion

Les problèmes de résolution d'emplois du temps universitaires sont de véritables problèmes dynamiques. Recommencer la résolution de zéro à chaque modification de

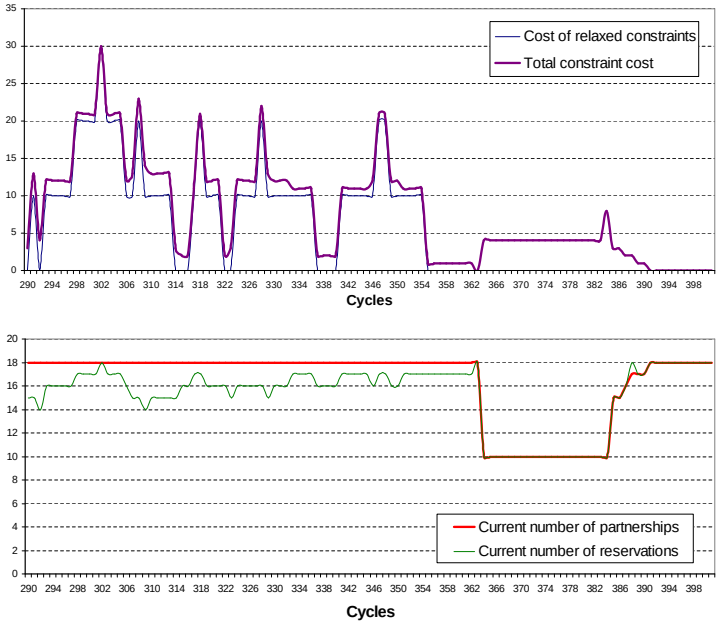

FIG. 3 – Variation du coût global (*en haut*) et des partenariats (*en bas*) durant la résolution d'un problème nécessitant une relaxation de contraintes avec suppression d'agent après la stabilisation du système.

contrainte peut être inefficace, voire infaisable en temps réel. Habituellement, l'objectif principal est d'obtenir un impact minimal sur la solution courante comme dans (Müller & Rudova, 2004) où le problème est résolu en introduisant un nouvel algorithme qui limite le nombre de perturbations additionnelles. Cambazard et al gèrent des problèmes dynamiques via une programmation par contraintes avec explications, plus particulièrement, de nouveaux opérateurs sont donnés pour effectuer une nouvelle propagation lorsqu'une contrainte est enlevée et que ses effets passés sont défaits (Cambazard *et al.*, 2004). Dans ETTO, dès qu'une contrainte est ajoutée ou supprimée pour un agent, ce dernier questionne ses réservations et son partenaire ; s'il juge qu'elles sont inconsistantes avec le nouvel état, il essaie de trouver une autre réservation ou un autre partenaire en explorant la grille et en suivant son comportement de base. L'intérêt est que le coût de la solution et le plus équitablement réparti possible. Si un nouvel agent est ajouté, il commence immédiatement à chercher un partenaire et une réservation, et s'il est supprimé alors toutes ses contraintes et réservations sont supprimées, ses frères et son partenaires sont prévenus. Le principal atout d'ETTO est que les modifications peuvent être faites sans arrêter la recherche pour une solution en progression. Les événements provenant des acteurs sont pris en charge à la volée. De plus, cette capacité à introduire des agents nous a permis de montrer que l'ajout d'agent surnuméraire apporte en efficacité.

Published in *7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, Plate-forme AFIA, Nice, mai 2005.

Mais ETTO possède des points faibles. Par exemple, résoudre des problèmes surcontraints n'est pas totalement efficace car si les agents ont trouvé une solution, ils continueront à explorer la grille pour trouver une solution plus pertinente. Comme les agents n'ont qu'une perception limitée de leur environnement, ils ne peuvent prendre en compte le coût global des contraintes pour arrêter leur recherche. Pour utiliser ETTO, nous considérons qu'il existe un oracle (l'humain responsable des emplois du temps) qui arrêtera le processus de résolution lorsque l'organisation répondra à ses besoins – avec un coût minimal de contrainte, par exemple. De plus, la recherche pour une cellule dans la grille n'est pas efficace car effectuée de manière aléatoire. Pour le moment, nous ne nous intéressons pas à l'efficacité, nous voulons juste montrer que notre approche par auto-organisation coopérative peut produire des résultats positifs. Néanmoins, la prochaine étape de notre travail est d'améliorer ETTO en fournissant la gestion coopérative de la mémoire des cellules afin d'explorer la grille plus efficacement.

Enfin, nous avons choisi un exemple simple pour appliquer ETTO. Une de nos perspectives est de se baser sur un problème plus complexe ou sur un cahier des charges comme celui donné par le réseau *Metaheuritics Network*[3]. Ceci nous permettra de comparer notre approche à d'autres, conceptuellement proches, comme les algorithmes génétiques, le recuit simulé (mais qui reste à un point de vue global sur le système et sa fonction) ou les algorithmes fourmis, intrinsèquement plus distribués.

5 Conclusion

Nous pensons que le caractère distribué inhérent au problème d'emplois du temps universitaires justifie l'approche par multi-agent pour le résoudre. Contrairement à d'autres travaux, avoir un protocole de négociation entre agent n'est pas la seule possibilité pour trouver une solution. Nous avons proposé une solution basée sur les systèmes multi-agents adaptatifs dans lesquels la coopération est un critère pour le changement des interactions entre agents afin de faire émerger la fonction globale du système. Ce type de programmation peut être assez efficace pour résoudre des problème complexes, peu ou mal spécifiés, et pour lesquels les concepteurs ne possèdent pas d'algorithme préétabli. Ceci est montré par les résultats préliminaires obtenus grâce au prototype d'ETTO aussi bien que par des travaux antérieurs sur la résolution de problèmes divers.

Remerciements

Nous tenons à remercier les acteurs d'ETTO et de son insertion dans ADELFE[4] : Carole Bernon, Valérie Camps Marie-Pierre Gleizes, François Bérenger et Sylvain Peyruqueou.

Merci aussi aux relecteurs pour leurs critiques avisées sur le fond et la forme qui nous ont permis d'améliorer la présentation de notre travail.

[3] `http ://www.metaheuristics.org`

[4] `http ://www.irit.fr/ADELFE`

Références

ALI S., ZIMMER R. & ELSTOB C. (1997). The question concerning emergence : Implication for Artificiality. In DUBOIS, D.M., Ed., *1^{st} CASYS'97 Conference*.

BISTARELLI S., FARGIER H., MANTANARI U., ROSSI F., SCHIEX T. & VERFAILLIE G. (1999). Semiring-based constraints CSPs and valued CSPs : frameworks, properties, and comparison. *Constraints : an International Journal*, **4**(3), 199–240.

BONABEAU E., THERAULAZ G., DENEUBOURG J.-L., ARON S. & CAMAZINE S. (1997). Self-organization in social insects. *Trends in Ecology and Evolution*, **12**, 188 –193.

CAMBAZARD H., DEMAZEAU F., JUSSIEN N. & DAVID P. (2004). Interactively Solving School Timetabling Problems using Extensions of Constraint Programming. In *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA*.

CAMPS V., GLEIZES M.-P. & GLIZE P. (1999). A Theory of Emergent Computation Based on Cooperative Self-Organization for Adaptive Artificial Systems. In *4^{th} European Congress of Systems Science, Valencia*.

CAPERA D., GEORGÉ J., GLEIZES M.-P. & GLIZE P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *1^{st} International TAPOCS Workshop at IEEE 12th WETICE*, p. 383 –388 : IEEE.

DECHTER, MEIRI & PEARL (1991). Temporal constraint networks. *Artificial Intelligence*, **49**, 61–95.

GEORGÉ J.-P., EDMONDS B. & GLIZE P. (2004). Making Self-Organising Adaptive Multiagent Systems Work. In F. BERGENTI, M.-P. GLEIZES & F. ZAMBONELLI, Eds., *Methodologies and Software Engineering for Agent Systems (Chapter 16)*, p. 321 –340 : Kluwer.

GOLDSTEIN J. (1999). Emergence as a Construct : History and Issues. *Journal of Complexity Issues in Organizations and Management*, **1**(1).

KOHONEN T. (2001). *Self-Organising Maps*. Springer-Verlag.

MÜLLER T. & RUDOVA H. (2004). Minimal Perturbation Problem in Course Timetabling. In *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA*.

SOCHA K., KNOWLES J. & SAMPELS M. (2002). A MAX-MIN Ant System for the University Timetabling Problem. In *Proceedings of 3^{rd} International Workshop on Ant Algorithms, ANTS'02*, volume 2463 of *LNCS*, p. 1 –13 : Springer-Verlag.

YOKOO M., DURFEE E., ISHIDA Y. & KUBAWARA K. (1998). The Distributed Constraint Satisfaction Problem : Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, **10**, 673–685.