ETTO : Emergent Timetabling by Cooperative Self-Organization

Gauthier Picard, Carole Bernon and Marie-Pierre Gleizes

IRIT, Université Paul Sabatier
F-31062 Toulouse Cedex, FRANCE
{picard,bernon,gleizes}@irit.fr
http://www.irit.fr/SMAC

Abstract. Cooperation is a means for multi-agent systems to function more efficiently and more adaptively. Cooperation can be viewed as a local criterion for agents to self-organize and then to perform a more adequate collective function. This paper mainly aims at showing that with only local rules based on cooperative attitude and without any global knowledge, a solution is provided by the system and local changes lead to global reorganization. This paper shows an application of cooperative behaviors to a dynamic distributed timetabling problem, ETTO, in which the constraint satisfaction is distributed among cooperative agents. This application has been prototyped and shows positive results on adaptation, robustness and efficiency of this approach.

1 Introduction

As a consequence of the growing complexity in the number of stakeholders or in the dynamics of software environments, artificial systems are more and more difficult to suitably design. The global function of those systems is often fuzzily specified but parts of the system are easily identifiable and local theories to model are well known. Such systems, qualified as *emergent* are studied by biologists and physicians for some years now. The two main properties of these systems are: the *irreductibility* of macro-theories to micro-theories [1] and the *self-organizing mechanisms* which are the origin of adaptivity and appearance of new emergent properties [8].

Kohonen networks or ant algorithms are two relevant examples of artificial transcriptions of self-organizing mechanisms [9, 3]. When considering less specific tasks, deciding when to reorganize in order to adapt to the environmental pressure, for reaching a global goal, needs equipping parts of the system with cognitive capabilities. Therefore parts become autonomous agents. As a response to this need of decision-making, the AMAS (*Adaptive Multi-Agent Systems*) approach proposes *cooperative attitude* as the local criterion used by agents to reorganize. Here, coooperation is not limited to ressource or task sharing, but is a behavioral guideline. Cooperation is viewed in a proscriptive way: agents have to locally change their way to interact when they are in *non cooperative situations* (or NCS). In AMAS, an agent is cooperative if it verifies the following meta-rules [5]:

c_{per}**:** perceived signals are understood without ambiguity,
c_{dec}**:** received information is useful for the agent's reasoning,
c_{act}**:** reasoning leads to useful actions toward other agents.

If an agent detects it is in a NCS ($\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$), it has to act to come back to a cooperative state and therefore to change the organization. The functional adequacy theorem [7] ensures that the function of the system is adequate – the system produces a function which is cooperative[1] for its environment – if every agent has such a cooperative behavior. Therefore, designing adaptive systems is equivalent to providing agents with a cooperative attitude and then ensuring the functional adequacy of the system.

The objective of this paper is to show that with only local rules based on cooperative attitude and without any global knowledge, a solution is provided by the system and local changes lead to global reorganization and then to a more adapted global function. In the next sections, this approach is illustrated by defining a cooperative behavior for agents having to dynamically solve an academic timetabling problem. Teachers and students groups have to find partners, time slots and rooms to give or to take some courses. Each actor has some constraints concerning its availabilities or required equipment. Moreover, a teacher can add or remove constraints at any time during the solving process via an adapted user interface. Such an application clearly needs adaptation and robustness. The system must be able to adapt to environmental disturbances (constraints modifications) and not to compute new solutions at each constraint changing. The correct organization has to emerge from actors interactions. This problem has been called ETTO, for *Emergent TimeTabling Organization*. To solve this problem, two kinds of agents have been identified and are presented in section 2. These agents respect several cooperation rules that are expounded in section 3. By now, the goal is not to be the most efficient, but to provide mechanisms to tackle changes with a minimal impact, as experiments show with results on adaptation and robustness of the AMAS approach in section 4. Section 5 discusses the approach and compares it to existing ones before concluding in section 6.

2 ETTO Agents

Two different classes of agents have been identified to tackle the ETTO problem: *Representative Agents* (RA) and *Booking Agents* (BA). The exploration of the solution space, a n-dimensional grid of cells, is delegated to Booking Agents. Each cell c_i of the grid is constrained (time slots, number of places, ...). All the constraints of a cell are regrouped in a set C_{c_i}. Cooperation between agents must lead to a correct organization by efficiently exploring the grid. This representation is shown in figure 1.

[1] non antinomic and non useless

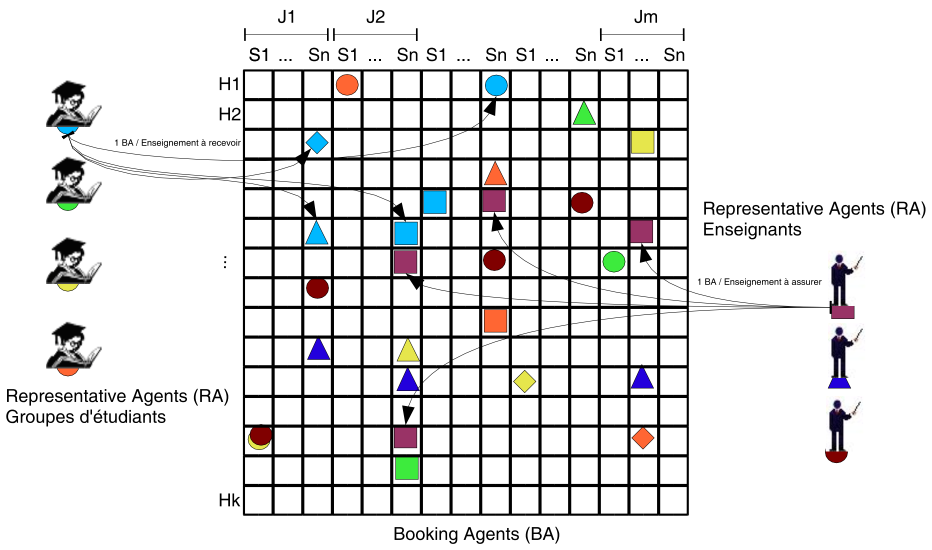

J1
J2
Jm
S1 ... Sn S1 ... Sn S1 ... Sn S1 ... Sn S1 ... Sn
H1
H2
Hk
1 BA / Enseignement à recevoir
Representative Agents (RA)
Groupes d'étudiants
Representative Agents (RA)
Enseignants
1 BA / Enseignement à assurer
Booking Agents (BA)

Fig. 1. Agents and environment in ETTO – each actor owns a Representative Agent and several Booking Agents to search partners and reservations in the n-dimensional grid to meet personal constraints.

2.1 Representative Agents

RAs are the interface between human actors (teachers or student groups) and the timetabling system. They own constraints (called *intrinsic constraints*) about availability, equipment requirements (projectors), or any other kind of personal constraint. To concurrently explore the possibilities of partnership and room reservation, those agents delegate exploration to Booking Agents. A RA (called *proxy*) creates as many BAs (called *delegates*) as it has courses to give (for teachers) or to take (for student groups), and randomly positions them on the schedule grid at the beginning of the solving. BAs from a same RA are called *brothers*. The task of a RA is simple: to warn its delegate BAs when its user adds or removes constraints and to inform all its delegate BAs when one of its delegate BAs produces new constraints (called *induced constraints*), to ensure coherency.

2.2 Booking Agents

BAs are the real self-organizing agents in ETTO. They have to reserve time slots and rooms and to find partners (student groups for teachers and vice versa) in accordance with constraints owned by their proxy.

In a cooperative situation, a BA, which is in a grid cell (i.e. a time slot in a room for a given day) books it and partners with another BA. But this nominal situation is not ensured at the beginning since BAs are randomly positioned. BAs then need to reorganize, i.e. to change their partnership and reservations, to find an adequate timetable. Such situations are NCS (see section 1). Therefore, BAs must be able to respond to NCS by respecting cooperation rules (see section 3).

Actions a BA can perform are simple: to partner (or unpartner) with another BA, to book (or unbook) a cell (by marking it with a virtual *post-it* with its address), to move to another cell[2], and to send messages to other agents *it knows*. A BA only knows its proxy and the BAs it encounters at runtime.

The life-cycle of a BA is a classical "perceive-decide-act" process as proposed by Capera et al [5]:

1. During the *perception* phase, the BA checks its messages (coming from other BAs or its proxy) and updates data about the cell (BAs in the cell, post-its, properties of the cell) in which it is positioned,
2. During the *decision* phase, the BA must choose the next action to perform to be as cooperative as possible, in accordance with the cooperation rules,
3. During the *action* phase, the BA performs the chosen action.

To perform its tasks, a BA ba_i has the following *local* properties, capabilities and knowledge:

– its current position in the grid ($cell(ba_i)$), which is the only cell the BA ba_i can see since it does not have a global knowledge of the whole grid,
– its current partner ($partnership(ba_i, ba_j)$ with $i \neq j$),
– its current reservation of the cell c_j ($reservation(ba_i, c_j)$ and $rCell(ba_i)$),
– its proxy ($proxy(ba_i)$),
– its search time ($time(ba_i)$) for a reservation, since it has no more reservation,
– the time slot of a cell ($slot(c_j)$),
– a limited memory of known BAs to send messages to (the set $knows(ba_i)$ or the predicate $knows(ba_i, ba_j)$), which is empty at the beginning of the solving and will be updated during the grid exploration,
– a set of intrinsic constraints (CI_{ba_i}) which are attached to the BA at its creation by its proxy RA,
– a set of constraints induced by its brothers (CB_{ba_i}) which are attached to and updated by its proxy RA when one of its brother reserves a cell to avoid ubiquity situations (two BAs of the same RA book the same time slot, for example),
– a set of constraints induced by its partner (CP_{ba_i}) which are attached to each partnership and updated when the partner changes its constraints to take into account the partner's preferences,
– a set of constraints induced by its reservation (CR_{ba_i}) to avoid partnering with a BA which is not available at certain time slots,
– the set of constraints from a first set which are non compatible with constraints of a second set ($nonCompatible(C_i, C_j) \subseteq C_i$) to process potential partners or cells to reserve,

[2] BAs do not know the whole grid, so moving to another cell implies defining the cells it knows from it.

– a function to weight constraints ($w(c_i) > 0$). The higher the weight is, the more difficult the constraint can be relaxed. A constraint c_i cannot be relaxed if $w(c_i) = +\infty$.

A macro, NC, is defined to simplify notations:

Definition 1. *The set of non compatible constraints between two BAs is* $NC_{ba_i,ba_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CR_{ba_i}, CI_{ba_j} \cup CB_{ba_j} \cup CR_{ba_j})$.

To determine the non compatible constraints between two BAs, the constraints coming from partners (CP) are not taken into account. In the same manner, for determining if a cell is compatible with a BA's constraints, constraints from the current reservation (CR) are not included:

Definition 2. *The set of non compatible constraints between a BA and a cell is* $NC_{ba_i,c_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CP_{ba_i}, C_{c_j})$.

By using NC, two constraint owners (BA or cell) can know if they are compatible:

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

Before starting the solving, there is no absolute way to decide what are the most difficult sub-problems to solve. Moreover, the difficulty degree could evolve due to the dynamic evolution of the problem description. Therefore, during the solving, each agent must be able to evaluate the difficulty it has to find a partner or a reservation. A BA ba_i can calculate the cost of a reservation of a cell c_j ($rCost(ba_i, c_j)$) and the cost of a partnership with another BA ba_j ($pCost(ba_i, ba_j)$) as following:

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

Dividing by the $time(ba_i)$ of search prioritizes the BA which is searching a cell for a long time. In fact, informally, it is cooperative to help agents having difficulties to find a position within the organization.

2.3 Basic Behavior

BAs have two orthogonal goals: *find a partner* and *find a reservation*. The main resolution algorithm is distributed among BAs and relies on the cooperation between agents. Solving is the result of dynamic interactions between distributed entities (BAs). As BAs have to reach two main individual goals, the nominal behavior they follow can be expressed in terms of the achievement of these goals, as shown in the algorithm 1.

During the perception phase, the BA checks its mailbox, in which other BAs can put messages about partnership requests or reservations. If these messages inform that its goals are reached (partnership and reservation), it moves to its

Algorithm 1 – Basic behavior for a BookingAgent.

```
while alive do
  processMessages()
  if partner AND reservation then              //reservation is optimal
    if rCost(ba_i,rCell(ba_i)) == 0 then
      moveTo(reservedCell)
    else         //analyze cell to find either partner or reservation
      processCurrentCell()
    endif
  else
    moveTo(nextCell);                     //choose another cell to explore
    addBAsToMemory();                 //memorize BAs which are in the cell
    processEncounteredBAs(); //verify whether they fit with constraints
    if NOT (reservation OR partner) then          //goals not reached
      processCurrentCell()                  //analyze the current cell
    endif
  endif
done
```

reserved cell only if this reservation is not too constrained. If the agent has relaxed some or if it lacks partner or reservation, it will explore the grid to find a (better) solution and analyze encountered BAs in memory and known cells, i.e. it verifies whether encountered BAs or cells better fit its constraints. Exploring the grid implies the capability for the agent to choose a next cell to explore. In the next experimentation, this is randomly done.

2.4 Constraint Management and Related Works

Actions may lead to add new induced constraints. For example, a BA which books a cell corresponding to a given hour at a given day warns its brothers, via its proxy, that this time slot is forbidden to avoid ubiquity situations. Conversely, if a BA unbooks a cell, it must inform its brothers. Therefore, a BA must process two kinds of constraints: intrinsic ones, which come from the actor its proxy RA represents, and induced ones, that come from its brother BAs. Of course, some problems may not have any solution without constraint relaxation. As a consequence, BAs must be able to affect priorities and weights to constraints as in fuzzy CSP or weighted CSP [2]. But, contrary to classical dynamic CSP [6], memory of previous states is sprayed within all the BAs which could be distributed within several servers. Finally, contrary to all these approaches, BAs only reason on a limited number of known BAs to find a good solution as in distributed CSP [16] or more accurately in distributed constraint optimization problems (DCOP) [12] that aim at optimizing (minimizing) the sum of relaxed constraints. Since BAs are agents, they do not have any global knowledge. Therefore, constraint optimization is shared by BAs, and the solution emerges from their local peer-to-peer interactions. Nevertheless, our approach remains different from above-mentioned ones, because the main objective is not to provide an algorithm that is sound, complete, and terminates, but to define local and robust mechanisms, able to implement a global solving. Similarly to applications of ant algorithms on scheduling problems [15], BAs alter their environment (the grid) with markers to indicate the cells they book and to constrain the other agents. The main difference with the usage of pheromone is the way the markers disappear. In the ant approach, markers evaporate with time. In our algorithm, markers are removed consequently to negotiation between BAs in booking conflict (see section 3.4). Finally, this work is close to local search based CSP approaches such as [11], but by using the agent paradigm to encapsulate constraints.

3 Cooperative Self-Organization Rules

The solving algorithm we propose is distributed among BAs and resides in cooperative self-organization. As said in the previous section, a nominal behavior of an agent is not sufficient to lead the collective to an adequate organization. BAs have to respect some cooperation rules to reach a correct global state (a pareto-optimal solution). Designing cooperative agents is then equivalent to implementing the cooperation meta-rules (see section 1). Five different situations for reorganization are identified. The two firsts do not respect the c_{dec} meta-rule of AMAS. The three next ones do not respect c_{act}. In this example, there is no c_{per} violation because all BA agents are identical and can understand each other.

The idea is to design these rules as *exceptions* in classical object programming, at the agent level and not at the instruction level. This concept really fits with the proscriptive approach proposed by [5]. As for exceptions, designers have to specify the condition of the exception throwing and the action to perform in the exception case. The following cooperation rules are then presented as condition-action pairs. Conditions are not exclusive. Nevertheless, a policy must be defined in the case of multiple NCS: from c_{dec} to c_{act}, for example.

3.1 Partnership Incompetence ($\neg c_{dec}$)

One of the goals a BA has to reach is to find a partner. If a BA ba_i encounters, in a cell, another BA ba_j it cannot partner with; ba_i is, using the AMAS terminology, incompetent [5]. For example, a BA representing a teacher's course meets another BA representing another teacher's course. As the only entity able to detect this partnership incompetence is the agent itself, this latter is the only one which changes the state of the organization by changing its position to encounter other more relevant BAs. Moreover, to enable a more efficient exploration of partnership possibilities, ba_i will memorize the location and the BAs known by ba_j for exchanging them during further encounters.

This cooperative self-organization rule can be summed up in the following table:

Name: **Partnership Incompetence** (for agent ba_i)

Condition:
$\exists j (j \neq i \wedge knows(ba_i, ba_j) \wedge (\neg compatible(ba_i, ba_j) \vee (pCost(ba_i, ba_j) \geq pCost(ba_i, partnership(ba_i)))))$

Action: `memorize(ba`$_i$`,knows(ba`$_j$`));move`

The $pCost$ comparison allows ba_i to decide if the potential new partner ba_j is less constraining than the current one.

3.2 Reservation Incompetence ($\neg c_{dec}$)

In the same manner than partnership incompetence, BAs must be able to change organization when their reservations are not relevant. This reservation incompetence NCS occurs when a BA ba_i occupies a cell which constraints do not fit its own constraints. For example, a BA representing a teacher's course is in a cell representing a room with not enough seats for this course. Then ba_i must move to explore the reservation possibility space.

Name: **Reservation Incompetence** (for agent ba_i)

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

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

To improve the exploration of the grid, a BA memorizes the cells in which this NCS occurs to share it during negotiation or to avoid it when moving, like in a tabu search for example.

3.3 Partnership Conflict ($\neg c_{act}$)

Situations during which a BA wants to partner with another partnered BA may append. The agent must react to this partnership conflict by partnering or by moving. In this case, the cooperation is directly embedded within the resolution action: the partnership will be performed with the agent that has more difficulties to find partners (by comparing the $pCost$). When it partners, a BA also unpartners with its previous partner and informs its previous partner and its proxy.

Name: **Partnership Conflict** (for Agent ba_i)

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

Action:

```
if (pCost(ba$_i$,ba$_j$) < pCost(ba$_i$,partnership(ba$_i$)))
   then partner(ba$_i$,ba$_j$)
   else move
```

When it partners, a BA must inform its previous partner and its proxy. Since this algorithm is distributed, the `partner(ba`$_i$`,ba`$_j$`)` action must be atomic (in the sense of critical section access), but is composed of the following instructions:

```
unpartner(ba$_i$,partnership(ba$_i$));
setPartner(ba$_i$,ba$_j$);
inform(partnership(ba$_i$));
inform(proxy(ba$_i$))
```

3.4 Reservation Conflict ($\neg c_{act}$)

As for partnership, reservation may lead to conflict: a BA wants to reserve an already booked cell. A reservation conflict can be specified as follows:

Name: **Reservation Conflict** (for Agent ba_i)

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

Action:

```
if (rCost(ba$_i$,cell(ba$_i$)) < rCost(ba$_i$,reservation(ba$_i$)))
   then book(ba$_i$,cell(ba$_i$))
   else move
```

When it books a cell, a BA must warn its previous partner and its proxy to inform not to book in the same time slot. Similarly to the `partner` action, the atomic `book(ba`$_i$`,cell(ba`$_i$`))` action is composed:

```
unbook(ba$_i$,reservation(ba$_i$));
setBook(ba$_i$,cell(ba$_i$));
inform(partnership(ba$_i$));
inform(proxy(ba$_i$))
```

3.5 Reservation Uselessness ($\neg c_{act}$)

In the case a BA is in the same cell than one of its *brothers*, reservation is useless. Therefore it can leave the cell without analyzing it, and its occupants, to find another more relevant one.

Name: **Reservation Uselessness** (for agent ba_i)
Condition: $cell(ba_i) = cell(partnership(ba_i))$
Action: `processEncounteredBAs();move`

Processing encountered BAs corresponds to verifying, in a limited memory list of BAs the agent has already encountered or another agent has shared during a negotiation, whether the agents can find a relevant partner with minimum partnership cost.

4 Prototyping and Experiments

To validate the algorithm we propose, an ETTO prototype were developed and several tests carried out to underline the influence of cardinality, the benefit for dynamic problem resolution and the robustness. Experiments are based on a French benchmark for the timetabling problem [3]. This requirements set is decomposed into four variants from simple problem solving without constraint relaxation to system openness by adding or removing agents and constraints at run time. For each of them, we proposed a solution – not unique in many cases.

4.1 Influence of Cardinality

Cooperation is a collective glue to enhance collective interactions. Therefore, it really becomes a relevant self-organization criterion in systems with a high cardinality. The figure 2 shows the evolution of solving time as a consequence of the growing number of BAs in the system. For these experiments, we keep the same exploration space size by increasing the number of cells in the grid proportionally to the number of agents. Only availability constraints are owned by teachers: one time slot per day is forbidden. Once the maximum reached (average 8 BAs), the number of cycles (during which every agent acts one time) decreases as the number of BAs increases. The time that varies the less is the real time. Therefore, it is the more relevant indicator of the solving time evolution. Beyond 32 BAs, it has a logarithmic curve. More BAs the system has, more efficient the solving is – if a solution exists.

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

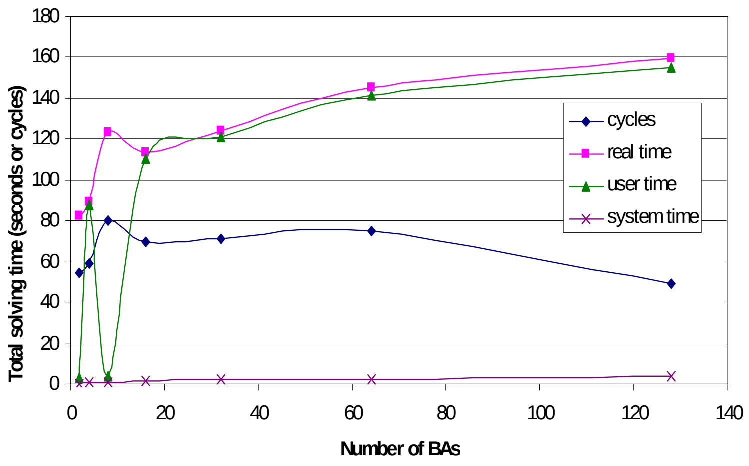


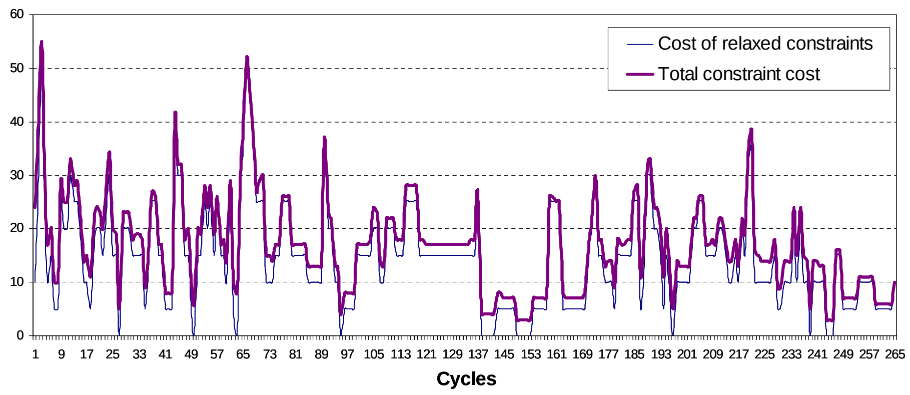
Fig. 2. Variation of solving time in terms of the number of BAs

4.2 Constraint Relaxing

In these experiments, agents must relax constraints to find a solution. The figure 3 shows the efficiency of ETTO solving, for a variant with 36 BAs requiring constraint relaxing. Reservations are set later than partnerships. ETTO found a solution with a constraint cost of 10 in 265 cycles. This cost represents the sum of all the constraints BAs had to relax, i.e. the sum of the weights of relaxed constraints. Nevertheless, the current prototype does not manage the cooperative slot sharing during negotiation and therefore, when a BA moves, it randomly chooses the next cell.

4.3 Dynamic Resolution

The two first experiments show the benefit of using cooperation to obtain an efficient timetable solving. A third serie of experiments tests the benefit in terms of robustness and dynamics. In these experiments, constraints become dynamic. Room or actors' availabilities may change at run time. Moreover, some agents can appear or disappear. By taking into account the chosen modeling, adding constraints is not different from adding agents that carry constraints. The figure 4 shows results on an experiment with initially 36 BAs. At cycle 364 – at the stabilization of the system – 8 BAs were removed, increasing the cost of relaxing constraints since each agent cannot find a relevant partner. 20 cycles later, 8 new agents with adequate constraints are plunged into the grid. The system only runs 7 cycles (from 384 to 391) to find an adequate organization with a null constraint cost.



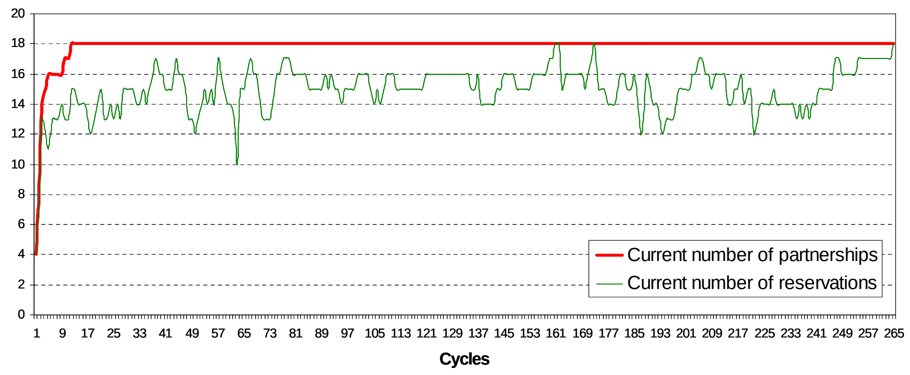
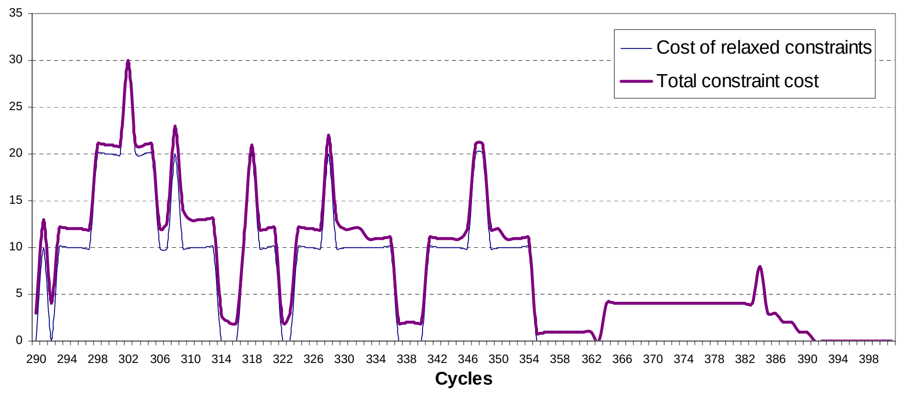


Fig. 3. Global constraint (*at top*) and partnership (*at bottom*) variations at run time for a solution requiring constraint relaxing.

5 Discussion

University timetabling problems in the real world are dynamic problems. Restarting from scratch each time a constraint is modified (added, removed) would not be efficient and some works are interested in this problem. Usually, the main objective is to have the smallest impact possible on the current solution as in [13] in which this is done by introducing a new search algorithm that limits the number of additional perturbations. In [4], explanations are used as well to handle dynamic problems, especially, new operators are given to re-propagate once a constraint removed and its past effects undone. In ETTO, as soon as a constraint is added or removed for an agent, this latter questions its reservations and its possible partnership; if it judges that they are inconsistent with its new state, it tries to find new ones by roaming the grid and applying its usual behaviour. If a new agent is added, it immediately begins searching for a partnership and if it is removed, then all its reservations and constraints are deleted from the system and its possible partner warned. The main feature of ETTO is that modifica-



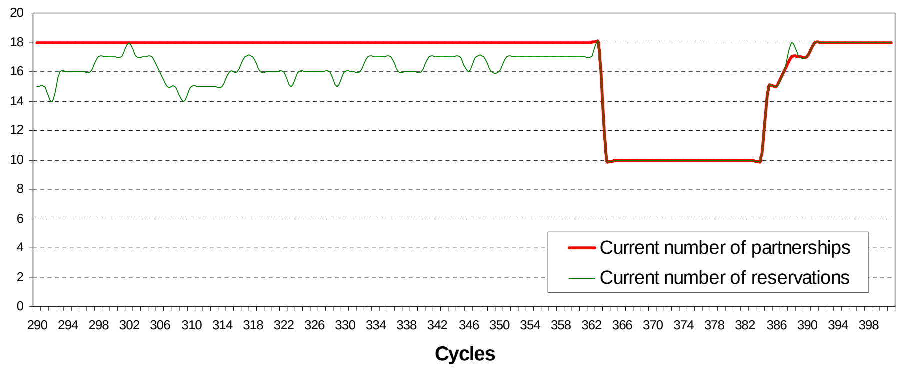


Fig. 4. Global constraint (*at top*) and partnership (*at bottom*) variations at run time with a removing of 8 agents after the system stabilization.

tions are then done *without stopping the search* for a solution while this latter is in progress, unexpected events are processed while actors are changing their constraints. Furthermore, this ability to insert agents has enabled us to show that adding supernumerary agents helps finding a solution and gives better results. This can be explained by the fact that the added agents can disrupt others which are satisfied with a solution that could be optimized. The main difference between our approach and the DCOP one [12], which considers the distributed optimization of the constraint cost, is the fact that the set of agents (BAs and RAs) are not totally ordered. RAs send constraints only to their child BAs and BAs send update data to only their father RA.

But ETTO has also weaknesses. For example, processing over-constrained problems is not fully efficient because even if agents have found a solution, they *continue to explore the grid* to find more relevant solutions. As agents have a *limited view* of the environment, they cannot take into account the global constraint cost to stop the exploration. To use ETTO, we consider it exists an oracle (a human manager) who will halt the solving process when the organization fits his requirements – a constraint minimum level, for example. The search for a cell in the grid is not efficient either because it is *randomly* made by an agent. For the time being, we were not interested by efficiency, we just wanted to show that our approach by self-organization is *feasible* and can produce positive results as it has been shown. The main positive result is that nothing in the behavior of agents makes assumption on how the solution timetable is obtained. This solution solely *emerges* from their local interactions. Nevertheless, a future step would be to enhance this search by *adding a limited memory* to agents concerning, for instance, the cells they visited in the past. We will also try other self-organizing mechanisms like the T-Man approach, which has been presented in the ESOA'05 Workshop [10].

Two problems are generally considered in most of the CSP approaches: a search problem in which a timetable that satisfies all the (hard/soft) constraints is first found and, then, an optimization problem which consists in minimizing an objective function that takes soft constraints into account. As the solving in ETTO is distributed among the agents, a global view of the solution does not exist and giving a global objective function is not possible. Therefore, optimization problems cannot be tackle in a global way. But, cooperation seems a relevant criterion anyway. In fact, we had applied this approach to more simple CSP (N queens, $N^2/2$ knights, etc) for which cooperation quickly provide solutions if there exist. In no-solution problems, systems quickly reach stable states with minimal constraint violation [14].

6 Conclusion

We think that the inherent distributed aspect of the timetabling problem explains a processing by a MAS. We proposed then a solution based on adaptive multi-agent systems in which cooperation is a local criterion for agents to change their interactions with others and to make the global, a priori unknown, function emerge from these interactions. This kind of programming can be efficient enough to solve problems that are complex, not well or incompletely specified and for which the designer does not possess a predefined algorithm. This is shown by the preliminary results obtained by ETTO as well as other previous works done in the solving problem domain.

We chose a rather simple example to apply ETTO, one perspective is to apply it to a more realistic timetabling problem or to a benchmark such as the one given by the *Metaheuristics Network*[4]. This will allow us to compare our results with other approaches such as genetic algorithms or simulated annealing.

References

1. S. Ali, R. Zimmer, and C. Elstob. The question concerning emergence : Implication for Artificiality. In Dubois, D.M., editor, *1st CASYS'97 Conference*, 1997.

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

2. S. Bistarelli, H. Fargier, U. Mantanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based constraints CSPs and valued CSPs: frameworks, properties, and comparison. *Constraints: an International Journal*, 4(3):199–240, 1999.
3. E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, S. Aron, and S. Camazine. Self-Organization in Social Insects. *Trends in Ecology and Evolution*, 12:188 –193, 1997.
4. H. Cambazard, F. Demazeau, N. Jussien, and P. David. 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*, 2004.
5. D. Capera, G. JP., M.-P. Gleizes, and P. Glize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *1^{st} International TAPOCS Workshop at IEEE 12th WETICE*, pages 383 –388. IEEE, 2003.
6. Dechter, Meiri, and Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
7. J.-P. Georgé, B. Edmonds, and P. Glize. Making Self-Organising Adaptive Multiagent Systems Work. In *Methodologies and Software Engineering for Agent Systems (Chapter 16)*, pages 321–340. Kluwer, 2004.
8. J. Goldstein. Emergence as a Construct : History and Issues. *Journal of Complexity Issues in Organizations and Management*, 1(1), 1999.
9. T. Kohonen. *Self-Organising Maps.* Springer-Verlag, 2001.
10. J. M. and O. Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Engineering Self-Organizing Applications – Third International Workshop (ESOA) at the Fourth International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'05), July 2005, Utrecht, Netherlands*, 2005.
11. S. Minton, M. Johnston, P. A., and P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:160–205, 1992.
12. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An Asynchronous Complete Method for Distributed Constraint Optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, pages 161–168, 2003.
13. T. Müller and H. Rudova. Minimal Perturbation Problem in Course Timetabling. In *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA*, 2004.
14. G. Picard and P. Glize. Model and Experiments of Local Decision Based on Cooperative Self-Organization. In *2^{nd} International Indian Conference on Artificial Intelligence (IICAI'05), 20-22 December 2005, Pune, India*, 2005.
15. K. Socha, J. Knowles, and M. Sampels. 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*, pages 1 –13. Springer-Verlag, 2002.
16. M. Yokoo, E. Durfee, Y. Ishida, and K. Kubawara. The Distributed Constraint Satisfaction Problem : Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.