Solving a Real-life Time Tabling and Transportation Problem Using Distributed CSP Techniques.

Gadi Solotorevsky and **Ehud Gudes**
Dept. of Mathematics and Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel
Email: {gadi,ehud}@cs.bgu.ac.il

Introduction

Many real-life problems in the domain of resource allocation and scheduling require solutions which are composed of several approaches or techniques. Often, complex problems are divided into sub-problems, sub-problems are solved by separate people (agents), who may use different techniques and expertise for solving them, and the sub-problems are combined later to yield a coherent, consistent solution. This last phase may involve negotiations with the various agents and requests for changes in their own solutions. A real-life example of the above problem is presented in this paper. The problem is the construction of a weekly timetable of nurses in several departments in a large Israeli hospital, and based on the departmental timetables, the construction of a transportation plan for all the nurses. This transportation plan tries to minimize the cost of transportation regardless of the departmental assignments; i.e., the number of vehicles sent and the distances they cover should be (approximately) minimal. It may happen that the agent responsible for transportation may ask for changes in the individual timetables, to avoid situations such as assigning a vehicle to bring in a single nurse from a far-away place. Obviously, this problem is a real-life instance of the distributed resource allocation problems.

The hospital requires that its departments maintain local control on the assignment of nurses; this ruled out the option of using a centralized algorithm to solve the whole problem. One alternative was to use a synchronized distributed algorithm (i.e., implementing a *standard CSP' algorithm*, in a distributed environment). In (Solotorevsky & Gudes 1996) we show that synchronized distributed algorithms are quite ineffective for performance, since this approach requires sending numerous messages, which slows the solution process. Moreover combining local control in a synchronized distributed algorithm is unnatural. Another alternative was to use *asynchronous algorithms* as the ones developed by Yokoo (Yokoo *et al.* 1992; Yokoo 1995) however these algorithms assume that each agent manages only one variable. It is possible to extend these algorithms to situations where each agent manages an entire sub-problem (e.g., a department of a hospital), however such extension poses many obstacles due to the non homogeneous difficulty of the sub-problems, and may greatly damage the overall performance of the system. In (Solotorevsky & Gudes 1996) we developed an approach to solve DCSPs (Distributed Constraint Satisfaction Problems) which was specially designed for situations in which each agent handles a complete sub problem, as opposite to a single variable. Furthermore, our approach takes advantage of the differences between the difficulties of the sub-problems. Our approach is based on a forward searching stage that is completely asynchronous and a backtracking stage that is semi-asynchronous. That is, the backtracking itself is done synchronously, but in the stages where the backtracking takes place all the agents which are not actively participating in the backtracking process, work asynchronously in searching for alternative solutions, solutions that will be available when the backtracking stage ends (for work on distributed scheduling not based on the CSP paradigm see (Sycara *et al.* 1991) and (Neiman & Lesser 1996)).

In (Solotorevsky & Gudes 1996) we tested our algorithms for solving DCSPs on random DCSPs with varying characteristics. In this paper we show how to apply and extend these algorithms to the nurses' timetabling and transportation problem (NTTT).

In the next section we describe in detail the nurses problem. In the third section the original algorithms are described and in the fourth section they are extended for the problem in hand. It should be noted that our problem is a real problem which is currently being solved partly by a knowledge-based program and partly by human experts. The last section discusses the suitability of our DCSP algorithms to this situation and compares their behavior to sequential algorithms.

The Nurses' Time Tabling and Transportation Problem.

A large hospital is composed of several departments; each department has its own staff of nurses. In each day there are 3 shifts: morning, evening, and night. The shifts have fixed starting and ending times that

are common to all the departments. The head nurse of each department makes the time table for the nurses of the department. Making the time table in a department consists of the assignment of nurses to shifts according to the requirements of the department and the personal constraints of the nurses. The requirements of the department specify how many nurses are needed in each shift, and if nurses with special skills are needed for a certain shift (see (Meisels, Gudes, & Solotorevsky 1997) for an extensive discussion on Employee Timetabling problems).

The nurses in the hospital come both from the city where the hospital is located and from several surrounding towns (which may be up to 60 Km. away). The hospital management rents transportation services for picking up and returning home the nurses that live in the surrounding towns. The transportation service is composed of 11 "lines", each line serving several surrounding towns. The transportation consists of taxis that may take up to 7 passengers. The hospital rents the taxis according to demands; i.e., the hospital may rent for a line more than one taxi at a certain time and none at another time. A taxi may go only to cities that belong to the same line. A taxi does not enter all the towns that belong to its line, but only those to/from which it actually carries passengers.

The hospital pays the transportation company per taxi (ignoring the number of passengers) according to the number of towns the taxi entered. For example, using two taxis each entering two towns is more expensive than using two cars but dividing the passengers so that each car enters one town. Using one taxi is always cheaper than using two taxis notwithstanding the number of towns visited by the taxi.

The scenario required by the hospital is first to solve the assignment according to the departments' and nurses' objectives and constraints. Afterwards, the proposed solution is checked to see if it fits some constraints about the transportation. If so, the nurses are divided into the lines trying to achieve a minimum cost. Otherwise the departments are asked to make some changes in their timetables and so on.

The problem of finding a distribution of the nurses (whose shifts assignment was already determined) between the lines and taxis that minimizes the cost of transportation is a classical optimization problem; therefore we will not deal with it in this paper. Our focus will be on the problem of finding the *schedule* of nurses that will satisfy both the needs of the departments and the transportation constraints.

One constraint posed by the hospital is that no taxi should take less than 4 passengers. The number of nurses served by each of the lines varies greatly. When the number of nurses served by a line is relatively large, the previous restriction will rarely rule out a solution proposed by the departments due to the assignment of nurses belonging to this line. However there is a line that serves only a reduced number of nurses (and these nurses may use only this line). It serves about 10 nurses. Therefore we should expect that the constraint about the minimal number of passengers in a taxi will rule out many timetables proposed by the departments.

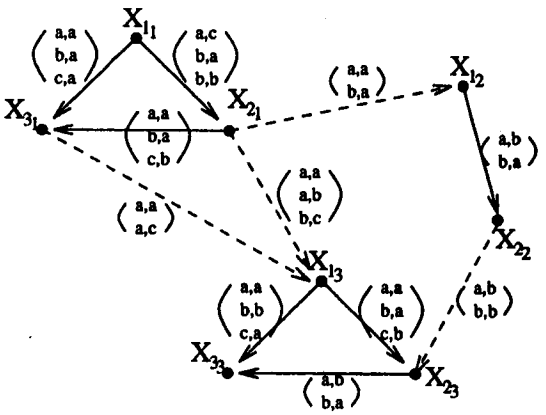

Figure 1: An explicit DCSP

Distributed Constraints Satisfaction Problem

The approach we use in this paper is based on the constraint satisfaction paradigm (Prosser, Conway, & Muller 1992; Sycara *et al.* 1991). A distributed CSP can be viewed as a set of constraint networks(CN), each CN being solved by a different agent, where the CNs are connected by constraints. A major assumption of our work is that checking constraints *inside* a distributed component, has a much lower cost than checking constraints *across* different components. The latter check involves some kind of *message passing* that the solving algorithm would like to minimize. The most relevant study of distributed CSPs has been done by Yokoo (Yokoo *et al.* 1992; Yokoo 1995) and Luo (Luo, Hendry, & Buchanan 1993). The basic difference between our approach and theirs is that while they assume usually a homogeneous network of nodes, our approach assumes a natural partition of the DCSP into relatively large components, and tries to take advantage of the differences between the various DCSP components.

In general, a DCSP may be represented in two ways. The *Explicit* representation is the original one, where variables in one component may be connected by a constraint to any other variable in the same or in a different component. In the *Canonical* representation, a new, central component is added. This component contains copies of all variables which are connected by inter-component constraints, such that solving the CSP of this central component guarantees that all global constraints are satisfied. The equivalence of the two representations can be shown easily. Figure 1 and 2 are an explicit and a canonical representation

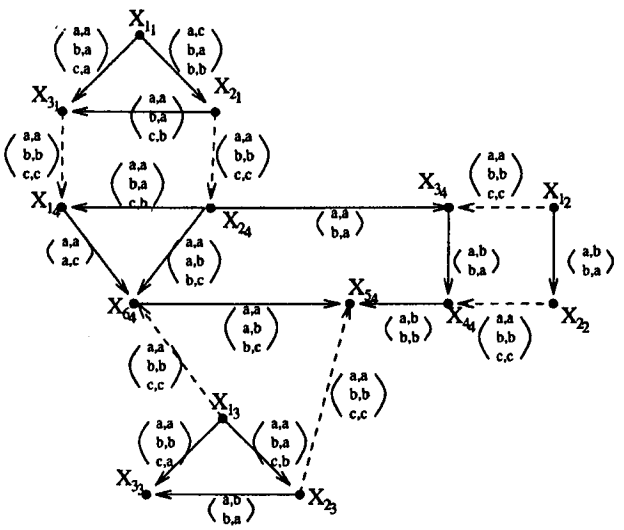

Figure 2: A canonical DCSP

of the same DCSP. In an explicit representation of the NTTT problem, each department is represented in one component and the transportation constraints are represented as constraints between the components, in a canonical representation of the NTTT problem a new component that maintains all the transportation constraints is added.

The formal definition of the canonical representation is as follows. A DCSP is defined to be a set of m groups $G_1, G_2 ..., G_m$ of variables and a mapping function M. For each $1 \leq i \leq m$ there exist in the i-th group, G_i, n_i variables $X_{1_i}, X_{2_i}, ..., X_{n_i}$ with domains $D_{1_i}, D_{2_i}, ..., D_{n_i}$. A **binary** constraint $R_{l_j k_i}$ between two variables X_{l_j}, X_{k_i} can be expressed as $R_{l_j k_i} \subseteq D_{l_j} \times D_{k_i}$. When $j = i$ the constraint is called *internal*, otherwise, when $j \neq i$, it is called *external*.

M is a function that maps the variables in the set G_m into variables on the other G sets. Each variable in the G_m set is *mapped to a single variable* in one of the other sets, and no two variables in group G_m are mapped to the same variable. A tuple P is a *solution to a DCSP network* iff:
1) All the binary constraints are satisfied in P.
2) Each variable that belongs to group G_m is assigned the same value in P as the variable on which it is mapped by M.

An **explicit** DCSP is a DCSP whose G_m group is empty, see Figure 1.

A **canonical DCSP** is a DCSP in which:
1) each external constraint includes exactly one node not in G_m and all other nodes are members of G_m.
2) For each internal constraint between nodes that all of them are mapped into nodes in G_m, there exists in G_m the same constraint between the mapped nodes.

An example of a canonical CSP is depicted in Figure 2 . The canonical representation of a DCSP problem is the basis for the two solution algorithms discussed next.

The Algorithms

The model of a DCSP in the present paper uses agents that are connected by a communication network (i.e., no common memory, just message passing). Since the overall goal is to find a global solution in the shortest time, we state the following goals for our multi-agent algorithms:

- Optimize the performance of the slowest agent, rather than optimizing each individual agent.
- Minimize the amount of backtracking each agent performs as a result of actions of other agents.

In (Solotorevsky & Gudes 1996) we presented two algorithms for solving DCSPs, CFPA (Central First Peripheral After), and PFCA (Peripheral First Central After), they can be summarized as follows:
1. CFPA. The first agent finds first a solution to the central component. It then broadcasts this solution to all the other agents. These agents search for solutions to their corresponding sub-problems in parallel. If all of them find a consistent solution to their subproblems, we are done. Otherwise, the central agent must backtrack and broadcasts a new solution to the peripheral components.
2. PFCA. Here, the peripheral agents search for solutions in parallel, and send their solution to the central agent. If the central agent can find a consistent solution we are done. Otherwise, the first agent that caused the failure is asked to backtrack, and send its new solution back to the central agent. The backtracking is done sequentially to assure completeness.

Algorithms CFPA and PFCA were designed for two opposite cases of DCSPs: a dominant central component seems natural for algorithm CFPA, while dominant peripheral components calls for algorithm PFCA. Note that these algorithms do not impose any specific strategy on the work of the internal agents, and those can use any suitable CSP strategy or even a Knowledge-based approach to solve their specific subproblem.

Although these algorithms seem quite simple, their implementation is not trivial and has many alternatives. For example in algorithm CFPA when the central component backtracks, the agent's problem is to correct a solution found by the central component by minimizing the number of changes required from solutions found so far. In (Solotorevsky & Gudes 1996) we used four low level procedures as the building blocks of CFPA and PFCA:

- **solve_internal**(G_i) is a procedure that finds a solution to the network G_i ignoring the external constraints.
- **propagate_external**(G_i) is a procedure that informs all the agents that are connected by con-

straints to G_i, about the values that were assigned to the variables of G_i.

- **update_propagate(G_i)** is a procedure that updates the agents that are connected by constraints to G_i, about changes in assignment of values to variables that were reported by G_i the last time that either propagate_external(G_i) or update_propagate(G_i) were used.
- **external_conflict_backtrack(G_i)** is a procedure that seeks an alternative solution for G_i, that is different from all the solutions found for G_i since the last call to solve_internal(G_i).

Note that much of the sophistication that may greatly affect the algorithms performance is hidden in the implementation of the four basic procedures, e.g., solve_internal may be implemented to use learning algorithms whenever the agent is idle.

In (Solotorevsky & Gudes 1996) the behavior of the proposed algorithms was tested by generating and solving a set of random DCSPs. The advantages of the two algorithms over the sequential CSP algorithms was clearly shown. In the following section we will discuss the use and adaptation of these algorithms for solving the NTTT problem.

Solving the Problem

The NTTT problem presents difficulties to both algorithms CFPA and PFCA. The difficulties derive from the fact that its quite easy to solve the central problem (transportation) separately, or the peripheral problems (timetable) separately, since each individual problem has many solutions; however very few of the combinations of these solutions are consistent with the transportation constraints. In more detail, we note that the zones of transportation can be roughly divided into two types. The first zone, serving only 10 nurses, is very constrained, therefore, the probability that a separate timetable created by the various departments will fulfill this constraint by chance is quite small! Thus, the first zone is more appropriate for algorithm CFPA. For the second zone that serves all the other lines, the central problem is not heavily constrained; therefore PFCA is more appropriate.

In order to overcome this problem we propose a new algorithm which is basically the adaptation and composition of the two basic algorithms. The original problem is first partitioned by a *binary partition* into two related DCSP sub-problems, then the following are applied:

1. Apply algorithm CFPA to the first sub-problem
2. Now use PFCA on the second sub-problem. If there is no solution, perform the backtracking on the components participating in the central component of the first sub-problem. Once the first sub-problem is free of conflicts there is a high probability that there are no conflicts in the second sub-problem either.

In terms of the NTTT problem, the first sub-problem is composed of the first zone and the components of the peripheral problems connected to this zone. The second sub-problem is the second zone and the rest of the peripheral components. The new algorithm is detailed and proven complete in the next section. [1]

Completeness of the Combined Algorithm

We first define the concept of a *Binary partition* of a canonical DCSP. Given a DCSP D let us define $\mathcal{N}(D)$ the set of nodes of D and $\mathcal{C}(D)$ the set of constraints in D. Given a *canonical representation* of a DCSP D and a partition of the variables of into two groups N_1 and N_2 let us define D' to be a DCSP that includes all the nodes and constraints in D and for each couple of variables (X_l, X_k) such that $X_l \in N_1$ $X_k \in N_2$ and neither of them is in G_m if exists a constraint $(X_k, X_l) \in \mathcal{C}(D)$, then we will add nodes X_{lm} and X_{km} to the G_m group of D' and identify constraints between X_k and X_{km}, and between X_l and X_{lm}.

Let us define a *binary partition* as a partition of the nodes and constraints of a DCSP into two parts, P_1, P_2, such that P_1 contains the set N_1 and P_2 contains the set N_2, that fulfills:

1. $X_k \in \mathcal{N}(D') \leftrightarrow \exists_{P_i \in P}(X_k \in \mathcal{N}(P_i))$. No node disappears and no node is added.
2. $X_k \in \mathcal{N}(D') \rightarrow \neg\exists_{P_i \in P, P_j \in P}(X_k \in \mathcal{N}(P_i) \wedge X_k \in \mathcal{N}(Pj) \wedge i \neq j)$. Each node belongs to one P_i.
3. $M(X_k) = X_l \wedge X_k \in \mathcal{N}(Pi) \rightarrow X_l \in \mathcal{N}(Pi)$. If X_k and X_l represent the same variable in the D', then they belong to the same P_i.
4. $\forall_{X_k \in \mathcal{N}(Pi), X_l \in \mathcal{N}(Pi)}(C_{kl} \in \mathcal{C}(P_i) \leftrightarrow C_{kl} \in \mathcal{C}(D'))$ i.e., all original constraints remain, and no new constraints are added.

In (Solotorevsky & Gudes 1996) we proved that CFPA and PFCA are *complete*, i.e. they terminate with a solution or with failure in finite time. Let us denote *CompleteExtAll* an algorithm that finds all the solutions of a DCSP that differ in values in the nodes in *Gm*. It is easy to modify CFPA and PFCA to be *CompleteExtAll* by as soon as a solution is found store it, then add a new constraint to the central component that states that the found solution is illegal, and continue.

Now, let P be a partition of a DCSP into two groups P_1 and P_2 , and let A_1 and A_2 be two CompleteExtAll algorithms for solving DCSPs. Algorithm CompCP (composition of CFPA and PFCA) is:

1. Apply A_1 to P_1: If no solution was found then halt with failure. Otherwise propagate the results to P_2.

[1] note that a practical result of this algorithm is that nurses on the first zone can coordinate their schedule, and it is likely that the system will fulfill their requests...

2. Apply A_2 to P_2 (with the domains reduced by the propagation) If no solution was found then undo the effects of the last propagation and goto 3. Otherwise halt with failure.

3. Use A_1 to find an alternative solution to P_1. If no alternative solution was found then halt with failure, otherwise propagate the results to P_2, and goto 2.

Theorem given a *binary partition* P of a DCSP D into two groups P_1 and P_2 then applying on them the *CompCP* algorithm is complete.

Proof:

Clearly D' has a solution if and only if D has a solution. Let $Sols_1$, and $Sols_2$ be the groups of all the solutions found by A_1 of P_1 and by A_2 of P_2, respectively.

¿From the construction of D' we get that Sol_1 (Sol_2) includes all the possible solutions that differs in values assigned to variables which have an *"external"* constraint to a variable in P_2 (P_1) (since they belong in D' to G_m, and A_1 and A_2 are CompleteExtAll). Therefore D has a solution if and only if there is a solution in $(s_1, s_2 \mid s_1 \in Sols_1 s_2 \in Sols_2)$

Both A_1 and A_2 are CompleteExtAll, therefore they will find the groups $Sols_1$ and $Sols_2$ in a finite time. Since both $Sols_1$ and $Sols_2$ are finite and were found in a finite time, then CompCP will terminate in a finite time. Therefore CompCP is Complete.

Note that in our algorithm the backtracking is done sequentially (although the backtracking time is used by other agents to find more solutions in parallel). In contrast the backtracking in (Yokoo 1995) and (Luo, Hendry, & Buchanan 1993) is asynchronous. On the surface its seems a disadvantage of our algorithm. However, our algorithm has important advantages on the asynchronous method. First, we use much fewer messages, and second we can incorporate more sophisticated backtracking methods such as *backjumping* easily. Finally, we deal easily with *non-binary constraints*. These types of constraints are very important in the type of problem we have on hand (e.g. limiting the number of nurses in a line is a non-binary constraint). The asynchronous algorithms require extensive changes to deal with these types of constraints in order to make them suitable to problems like the NTTT problem.

Experimental Evaluations

Our experiments with the different algorithms are summarized in table 1. We tested the behavior of CFPA, PFCA, CompCP, and two versions of sequential forward constraint checking with failure directed back jumping (FC-BJ), the first regular FC-BJ, and the second FC-BJ in which the nurses that belong to the "difficult line" are assigned first (FC-BJ-L-1st). We tested the algorithms on two versions of the NTTT problem, a full version that included 10 departments, 20 nurses in each department, half of them from surrounding towns, each nurse can work up to five shifts per week, and each department requires about 100 weekly assignments. The second set of tests was on a reduced version of the problem with only 4 departments, 7 nurses in each department, each nurse works 2 shifts per week, and each department required about 20 weekly assignments.

	Toy Problem		Real Problem	
Algorithm	Messages	MNCC	Messages	MNCC
PFCA	123054	555517	stopped	stopped
CFPA	128	16336	stopped	stopped
FC-BJ	-	553964	stopped	stopped
FC-BJ-L-1st	-	525	-	150000
CompCP	8	148	20	15102

Table 1: Applying the different algorithms to the NTTT problem

The results are presented in table 1, the performance is measured both in terms of messages needed, and in terms of the *maximal* number of consistency checks (MNCC). We defined MNCC to be the sum of all the consistency checks that are performed in the sequential intervals plus the sum of the maximal number of consistency checks that are performed by one of the agents in each parallel interval. Note that by assuming that all the agents have a common clock, and that each internal constraint check takes one time unit we get an equivalence between the MNCC measurement and the time cycles measurement which is used by Yokoo (Yokoo 1995).

Algorithms that failed to solve the problem in 2 hours were halted (stopped). It is clear from the table that algorithms CFPA and PFCA alone can not deal with the NTTT problem however applying CompCP gives very good results, much better than the sequential algorithms, specially for the full size problem. It is also interesting to see that applying FC_BJ without dealing first with the "difficult line" is very inefficient.

Note that we did not measure the time required for solving the problems because time measurements are highly implementation dependent. Furthermore, in a distributed implementation, time measurements are highly dependent on the network architecture (e.g., LAN or WAN) and on its load. (Note that performing 15102 MNCC and 20 messages, may on many architectures take more time than performing 150000 constraints' checks in a centralized environment.) Our aim is to show that when a problem *requires* the use of a distributed solution method (as the hospital management demanded) then applying an appropriate distributed algorithm for the problem, enables its solution with a reasonable amount of work.

Discussion

The efficiency of applying CompCP to a problem depends on the capability to identify tightly and loosely constrained zones in the central component. When, in

a problem, the central component is uniformly difficult then PFCA or CFPA should be preferred to CompCP.

One can think about a general partitioning algorithm which attempts to identify the difficult component of the problem. However, in the NTTT problem, as well as in many other real life resource allocation and scheduling problems, many of the constraints are naturally represented in a functional form, and not in an explicit form (the reason for this is that an explicit representation may take exponential space). When the constraints are not given in an explicit form, finding the difficult regions of the central component automatically, without using some knowledge about the problem, doesn't seem feasible, (since it is basically equivalent to finding the number of failures due to these constraints over the total number of constraint failures for all possible solutions!), that is the reason why we need to use specific knowledge on the problem in order to find the partition of the central component. Note that the use of such knowledge was also mandatory for the sequential algorithm (FC-CBJ-L1), since the algorithm FC-CBJ which did not use knowledge specific variable ordering failed to solve the real problem in a reasonable amount of time.

To generalize on this, we claim that a DCSP problem of this nature, where part of the central component is much more difficult than the rest, and where the constraints are stated functionally, will usually require domain knowledge to identify the difficult part, and then the application of CompCP is quite obvious.

Conclusions

In this paper we investigated a real-life resource allocation problem - the timetabling and transportation of nurses in a large hospital. The solution methodology is based on the algorithms developed earlier for solving distributed constraint satisfaction problems. In the previous section we saw that this methodology enabled an efficient solution of the problem. The reason was the differences existing between solving the central problem vs. solving the peripheral problem, these differences are of great importance in our algorithms.

Other reasons for using our methodology are the possibility of preserving local control and reasoning, and the ability of incrementally constructing the overall system. The latter point is of particular importance in our case. In several departments, the timetabling is still done manually where only the results are encoded into input to the transportation part. In others, where the knowledge-based approach is used, the local expert wants complete control, even with the option of manually changing the schedule and violating some of the constraints. All these entails a methodology which gives as much local autonomy as possible to the individual agents, and does not force a particular algorithm on them. Note that even though preserving local control and using independent internal strategies are natural properties of the distributed AI approach, other approaches to distributed solving of CSP do not maintain them. This makes our methodology quite different from the uniform methodology for solving distributed CSP problems advocated for example by Yokoo (Yokoo 1995).

As was shown, the solution of our problem required the combination of the CFPA and PFCA algorithms. This is needed, because there are quite large differences between the difficulty of solution of the various peripheral problems. We believe that such composition would be useful in many problems where there are great differences between various components of a distributed CSP problem, and there is domain specific knowledge identifying these differences.

References

Luo, Q. Y.; Hendry, P. G.; and Buchanan, J. T. 1993. Heuristic search for distributed constraint satisfaction problems. Research Report KEG-6-93, University of Strathclyde.

Meisels, A.; Gudes, E.; and Solotorevsky, G. 1997. Combining rules and constraints for employee timetabling. *Intern. Jou. Intell. Sys* 12(6).

Neiman, D. E., and Lesser, V. R. 1996. a cooperative repair method for distributed scheduling system. In *Proc. of the Third Inter. Conf. on Artificial Intelligence Planning Systems*, 166–173.

Prosser, P.; Conway, C.; and Muller, M. 1992. A constraint maintenance system for the distributed resource allocation problem. *Intelligent Systems Engineering* 76–83.

Solotorevsky, G., and Gudes, E. 1996. Algorithms for solving distributed constraint satisfaction problems (dcsps). In *Proc. of the Third Inter. Conf. on Artificial Intelligence Planning Systems*, 191–198.

Sycara, K.; Roth, F.; Sadeh, N.; and Fox, M. 1991. Resource allocation in distributed factory scheduling. *IEEE Expert* 29–40.

Yokoo, M.; Durfee, E.; Ishida, T.; K.; and Kuwabara. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *IEEE Intern. Conf. Distrb. Comp. Sys.*, 614 - 621.

Yokoo, M. 1995. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proc. 1st Intrnat. Conf. on Const. Progr.*, 88 - 102.