

Distributed personnel scheduling—negotiation among scheduling agents

Eliezer Kaplansky · Amnon Meisels

Published online: 17 July 2007
© Springer Science+Business Media, LLC 2007

Abstract This paper introduces a model for *Distributed Employee Timetabling Problems* (DisETPs) and proposes a general architecture for solving DisETPs by using a *Multi Agent System (MAS)* paradigm. The architecture is composed of a set of autonomous software *Scheduling Agents (SAs)* that solve the *Employee Timetabling Problems (ETP)* for each department. Each agent has its own local ETP problem and its own goals. The Scheduling Agents must coordinate their local solution with the other agents in order to achieve a global solution for the whole organization that yields a better result with respect to the organization's global targets. To achieve a coherent and consistent global solution, the SAs make use of a sophisticated negotiation protocol among scheduling agents that always ends in an agreement (not ensured to be optimal). The main functionalities of this protocol are agent to agent relation definition, a mechanism to approve a chain of *Request for Changes* and an electronic marketplace for bidding on preferred common time slots.

Experimental analysis of the implemented *Multi Agent System* for the Soroka medical center is presented. The results of our study indicate that the proposed framework has the potential to reduce the cost of transportation for the nurses that traveling to and from the hospital.

Keywords Multi agent system · Employee timetabling · Automatic negotiation

1 Introduction

In the commercial world of today, many organizations strive to reduce their costs while maintaining a high degree of job satisfaction. One common goal is to improve the quality of timetabling for the organization's employees. This trend pushes the individual requests into a much higher priority.

E. Kaplansky (✉) · A. Meisels
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84-105, Israel
e-mail: kaplan_e@cs.bgu.ac.il

A. Meisels
e-mail: am@cs.bgu.ac.il

Timetabling Problems (TTPs) involve an organization with a set of tasks that need to be fulfilled by a set of resources. In *Employee Timetabling Problems (ETPs)*, the resources are employees with their own qualifications, constraints and preferences. The organization enforces overall regulations and attempts to achieve some global objectives such as lowering the overall cost.

Many real world timetabling problems are composed of organizational parts that need to timetable their activities in an independent way, while adhering to some global constraints. Employees need resources to execute their tasks. Many of the resources are shared by several parts of the organization. In universities, for example, the classrooms are shared by several departments of the university. We term these resources *Shared resources (ShRes)*. In order to avoid conflicts and to make an effective use of these shared resources, the timetables of all the parts must be combined to yield a coherent compatible solution. To achieve this goal, negotiations among multiple agents and requests for changes in their local solutions are needed. The overall problem becomes a *Distributed Timetabling Problem (DisTTP)*. When the local problem is employee timetabling, the overall problem becomes a *Distributed Employee Timetabling Problem (DisETP)*.

An example for a real-life distributed timetabling problem is the construction of a weekly timetable of nurses in all the wards of a large hospital. Each ward needs a weekly timetable for its nurses, that satisfies the shift and task requirements while striving to satisfy, as much as possible, the nurses' personal preferences. The generation of a weekly schedule for the nurses in a ward is typically performed by each ward independently.

Most hospital managements provide some kind of transportation services for the nurses working in the hospital. The *Shared Resources* of the hospital's DisETP are seats in the transportation lines for the hospital's nurses. Based on the timetables of all wards, a transportation plan for all the nurses must be constructed. Since the number of vehicles sent and the number of stops on each line are limited by hospital saving policies, such a transportation plan generates additional constraints between the weekly schedules of the wards.

The weekly timetable of each ward is constructed locally by a timetabling solver program. The independent solvers, or scheduler, of the different wards are termed *Scheduling Agents (SAs)*. The constraints among these timetables, imposed by the limits on transportation seats, have to be negotiated among the scheduling agents of the wards.

Recently, the AI community has shown an increasing interest in solving distributed problems by using the multi agents paradigm. *Multi-Agent System (MAS)* technology has become a common paradigm for modeling, designing, and implementing software solutions. Agents are sophisticated computer programs that act autonomously on behalf of their users, across distributed environments. A Multi-Agent System is a loosely coupled network of software agents that interact in order to solve a global problem that is beyond the individual capacities or knowledge of each agent. The distributed architecture proposed in this paper is in line with the irreversible trend toward using the available distributed computation resources more effectively. This need makes the current approach of a global, centralized, search engine that runs on a central computer less and less adequate.

This paper focuses on real world problems that by their nature need to be solved by distributed software agents. Each agent is responsible for solving a single, local, complex timetabling problem. The local solutions are combined by the communication protocol of the network of the scheduling agents to yield a coherent, globally consistent solution. To perform this operation effectively, when many agents are involved, requires a new framework that allows the agents to engage in negotiations. This framework include a protocol that enable an agent to issue a *requests for changes* in other agents' solutions.

The paper proposes a new model for solving distributed ETPs. The proposed inter agent negotiation protocol allows the scheduling agents to reach a global agreement by cycling

through three alternative negotiation stages. The first stage (the *Local Stage*) focuses on search for local solutions of the timetabling problem of each scheduling agent. Solving the local ETP of each ward is a complex timetabling problem that has been addressed in previous work (Grobner and Wilke 2001; Meisels and Schaerf 2003). The goal of the second stage (the *Global Stage*) is a feasible global timetable, free of the global resources conflicts. To achieve this goal, the *Resource Agents* and the *SAs* cooperate in a distributed search process and agree on changes of local timetables in order to achieve global resource compatibility. The distributed search algorithm is defined in the form of a sophisticated and well defined negotiation protocol among scheduling agents and resource agents that always ends in an agreement (not ensured to be optimal). This protocol enables the *SAs* to agree on a dynamic order for processing their local timetables and to submit their requests for changes in an unsynchronized way. In the third stage (the *Bid Stage*) the agents focus on improving the quality of their local solution while maintaining a feasible global solution.

Experimental analysis of the implemented MAS system at Soroka, Israel's second largest medical center, with a staff of more than 4000, shows that the proposed framework can help the hospital to significantly reduce the cost of transportation for its nurses.

This paper is organized into four parts:

Part I—Problem definition: The first part of the paper (Sect. 2) describes the Distributed Employee Timetabling Problem (DisETP) that is at the center of the paper. This includes a description of the local version of ETP, its constraints and its Constraint Satisfaction Problems (CSP) Model. In Sect. 2.4 a well known real life example of an ETP, the weekly timetable of nurses in a hospital is presented. Based on this example, a distributed CSP model for this distributed employee problem is described.

Part II—Solving the problem of each agent: Sect. 3 investigates search methods for solving the local ETP of each agent, where constraints are added or removed dynamically. These methods deal with *Dynamic timetabling problems* of the *SAs*. Section 3.2 describe the use we make with Gira to solve the local ETP. Gira *General Iterative Restart Algorithm* is a general solving method for CSPs that uses restart and randomization techniques.

Part III—Framework for Scheduling Agents: A framework for scheduling agents is described in Sect. 4. This include a simple architecture for solving small scale distributed personnel scheduling problems. This architecture is an instance of a *Multi Agent System* (MAS) paradigm that make use of a *Central Resource Agent*. Solving methods to deal *Open Constraint Satisfaction Problems* (OpenCSP) of the resource agents is described. For real world large scale ETPs, a more sophisticated architecture is needed (Sect. 5). This architecture makes uses of an inter-agent negotiation protocol for achieving a globally consistent timetable. In the end of the paper, in Sect. 7) a full scale negotiation protocol is proposed for future work.

Part IV—Case study—DisETP at a large hospital: A detailed description of a case study of DisETP is given in the last part of the paper. Section 6 describes the implementation of a DisETP system at SOROKA, Israel's second largest medical center. Algorithmic solutions for the transportation agent are described and results from experimental investigation that compares their behavior are shown.

2 Distributed employee timetabling

2.1 Employee timetabling problems

Employee timetabling problems (ETP) involve an organization with a set of tasks that need to be fulfilled by a set of employees. Each employee has his own qualifications and constraints.

The organization has regulations that impose constraints on the assignment of an employee to a task. The goal is to find an assignment to all tasks in shifts with fixed start and end times, such that all the constraints are satisfied.

A *task*, in the context of ETPs, is performed during a predefined time period called a *shift*. Shifts are fixed in time (e.g., a *morning shift*) and the term *timetabling employees* refers to a process of *assigning employees* to tasks in shifts. There can be many shifts in one day, possibly overlapping in time. The structure of shifts forms an immediate generalization to the common family of ETPs in which the basic time-slots are the days of the week (cf. Kragelund 1997; Caseau et al. 1993).

The ETPs that we consider are based on the concept of assigning employees to tasks in shifts with fixed start and end times. There are m employees E_1, \dots, E_m , n shifts S_1, \dots, S_n , and t tasks T_1, \dots, T_t . We need to find a 3-dimensional binary matrix $X_{m \times n \times t}$, such that $x_{ijk} = 1$ if employee E_i is assigned to task T_k in shift S_j .

Definition 1 A *General Employee Timetable* on the finite sets (E, S, T) is defined as a mapping $t : E \times S \rightarrow T$, where $E = \{E_1, E_2, \dots, E_m\}$ is a set of *employees*, $S = \{S_1, \dots, S_n\}$ is a set of time slots, and $T = \{T_1, \dots, T_t\}$ is a set of *tasks*.

2.2 Local constraints of ETPs

The constraints of the employee timetabling problem can be grouped into the following five sets:

Requirements: Each shift S_j is composed of a number of tasks, some of them multiple times. An employee is needed to be assigned for each task belonging to S_j . A non-negative integer matrix $R_{n \times t}$, called *Requirements* matrix is given, such that R_{jk} denotes the number of occurrences of task T_k in shift S_j , which corresponds to the exact number of distinct employees that must be assigned to task T_k in shift S_j .

Ability: Each employee has qualifications that enable him to fulfill certain types of tasks; that is, each employee E_i has a set of tasks $\{T_{i1}, \dots, T_{ir}\}$ that E_i can be assigned to. The *qualification* matrix is a binary matrix $Q_{m \times t}$ such that $Q_{ik} = 1$ if employee E_i qualifies for task T_k , $Q_{ik} = 0$ otherwise.

Availability: There are personal preferences of employees, which restrict them to be assigned only to subsets of the shifts. These constraints are represented by a binary matrix of *availabilities* $A_{m \times n}$, where $A_{ij} = 1$ if employee E_i is *available* for shift S_j and $A_{ij} = 0$ otherwise.

Conflicts: Obviously, an employee cannot be assigned to more than one task in the same shift. In addition, employees cannot be assigned to two shifts that are in *conflict* with each other. Sources of conflicts could be different: overlap in time, consecutive, or combinations that are forbidden by organizational rules. Conflicts may vary for different employees (because of different contracts) and are described by a 3-dimensional binary *Conflict* matrix $C_{n \times n \times m}$, such that if $c_{j_1 j_2 i} = 0$, then employee E_i cannot be assigned to both shifts S_{j_1} and S_{j_2} .

Workload: There is an upper and lower limit on the number of tasks that each employee can be assigned to, per schedule. There are actually a set of limits, because employees can be assigned to a limited total number of tasks per schedule and also to a limited (smaller) number of specific assignments. We therefore define a set of shift sets G_1, \dots, G_s , each one grouping shifts of a specific kind. Then, we define two integer-valued matrices $V_{m \times s}$ and $W_{m \times s}$ such that employee E_i must be assigned to at least V_{ik} and at most W_{ik} shifts of group G_k .

The problem that is at the focus of this paper is to find *any* assignment that satisfies all of the above constraints. The corresponding mathematical formulation is the following.

find x_{ijk} such that

$$\sum_{i=1}^m x_{ijk} = R_{jk} \quad (j = 1..n; k = 1..p), \quad (1)$$

$$x_{ijk} \leq Q_{ik} \quad (i = 1..m; j = 1..n; k = 1..p), \quad (2)$$

$$x_{ijk} \leq A_{ij} \quad (i = 1..m; j = 1..n; k = 1..p), \quad (3)$$

$$\sum_{k=1}^p x_{ij_1k} \cdot \sum_{k=1}^p x_{ij_2k} \leq c_{j_1j_2i} \quad (i = 1..m; j_1, j_2 = 1..n), \quad (4)$$

$$V_{ih} \leq \sum_{j \in G_h} \sum_{k=1}^p x_{ijk} \leq W_{ih} \quad (i = 1..m; h = 1..s). \quad (5)$$

Although our experiments that are presented in this paper are limited to cases of hard constraints, our model allows the articulation of soft constraints according to the following types.

Uniform load: Uniform load of assignments among (certain groups of) employees. The objective of uniform distribution of load among employees is related to specific groups of assignments and is corrected for the differences in minimal and maximal limits between employees.

Spreading: Preferred spacing of (more difficult) assignments, over the time span of the schedule. Objectives for preferred spacings between certain assignments arise naturally in many situations. When a nurse is assigned two `Night_Shifts` per weekly schedule, for example, it is considered much better to have them spread evenly over the week (i.e., at least two free nights in between).

Preferences: Granting employees their personal preferences. A general consideration of preferences of employees, for assignments to different types of shifts or times can be quite complex, involving multiple priorities for shifts (differing among employees). In the present ETP model there is only one type of preference and employees can attach this preference to any shift they wish. The objective for the optimization problem counts all the assigned preferences, but, normalizes the number of assigned preferences per employee by using the individual minimal and maximal work load.

For recent work in Employee Timetabling Problem see Schaerf and Meisels (1999), Kragelund and Kabel (1998), Burke et al. (1998, 2001).

2.3 A CSP model for scheduling agents

In the last decade, *Constraints Satisfaction Problem* (CSP) techniques have become the method of choice for modeling many types of optimization problems, in particular, those involving heterogeneous constraints and combinatorial search.

Constraints networks (CN), which are also called constraint satisfaction problems—CSPs, require the assignment of values to variables satisfying a set of constraints (cf. Tsang 1993; Dechter 2003 for introduction to CSPs). A constraint network is composed of a set of variables X_1, X_2, \dots, X_n , with domains of values for each variable D_1, D_2, \dots, D_n , and

a set of constraints among the variables. Constraints or *relations* R of a CN are subsets of the Cartesian product of the domains of the constrained variables $R \subseteq D_1 \times D_2 \times \dots \times D_n$. A *binary constraint* R_{ij} between variables X_i and X_j is a subset of the Cartesian product of their domains $R_{ij} \subseteq D_i \times D_j$. The case where $i = j$ denotes a *unary constraint* on X_i . A *solution* P to a binary CSP is an n -tuple (i.e., assignments of values to all variables) that satisfies all the constraints:

$$P \in \{(x_1, x_2, \dots, x_n) \mid \forall_{ij} x_i \in D_i \wedge (x_i, x_j) \in R_{ij}\}.$$

The three-dimensional binary matrix that was introduced for representing solutions in Sect. 2.2 leads to inefficient constraints representation due to the large number of variables involved. In a former paper, an alternative method to formalize the ETP model into a CN was proposed (Schaerf and Meisels 1999; Meisels and Schaerf 2003). The variables of the constraint network of ETPs are shift-task pairs $\langle S_j, T_k \rangle$. There is a pair of the form $\langle S_j, T_k \rangle$ for every required task in every shift S_j . Note that there are in general multiple such pairs, corresponding to needs of more than one employee in some tasks of any specific shift. Let us denote these variables $Y_{jk1}, Y_{jk2}, \dots, Y_{jkR_{jk}}$. The values that need to be assigned to the variables are employees. Every employee that has the ability to be assigned to the task T_k , will initially be in the domain of all variables of the form Y_{jkr} .

In this form, the number of variables of the constraint network equals the required number of assignments in the ETP. The unavailabilities of employees, arising from personal constraints, that forbid the assignment of employee E_i to shift S_j for example, are *unary constraints* (cf. Tsang 1993). The effect of the above unary constraint is to remove the value E_i from the domain of values of variables Y_{jkr} . The above representation accommodates well the requirements of the problems, their abilities and unavailabilities. In fact, the representation of requirements is concise, since the variables include only the relevant shift-task pairs and no counting is needed in order to check the requirements.

The constraints of ETPs fall into two main categories: binary constraints, which we called *conflicts* in the first part of this section, and non binary constraints of the *Workload* type (limiting the number of some set of assignments). Conflict constraints forbid the assignment of certain employees to two conflicting shifts. In the CSP literature, binary constraints are usually represented explicitly in matrix form (cf. Prosser 1993; Tsang 1993; Dechter 2003). The binary part of any constraint network can be characterized by the density and tightness of these constraints (cf. Prosser 1994). For real world ETPs the density of constraints is not high (see Meisels and Lusternik 1997). In addition, the constraint networks of real world problems tend to be non uniform and to have a wide range of domain sizes (Meisels and Lusternik 1997).

Limits on the number of assignments of certain values (i.e., *employees*) to a set of variables (*shift-task* pairs), are non binary constraints on sets of variables of *CN_ETPs*. These, workload, constraints require suitable representations and ordering heuristics for their processing. Such representations and heuristics, within the context of constraints networks of TTPs, were presented in a former study (Solotorevsky et al. 1998).

2.4 Nurses timetabling and transportation

The timetabling of nurses in wards of a large hospital is a well known, real life, example of the Employee Timetable Problem. Several artificial intelligence techniques for nurse scheduling were investigated. These include declarative approaches, constraints programming, expert systems, etc. Many of the most recent paper tackle the problem with meta-

heuristic approaches (Burke et al. 1998, 2001; De Causmaecker and Vanden Berghe 2002; Petrovic et al. 2002). However the nurse scheduling problem can be more naturally modeled as CSP as described in the previous section.

A large hospital is composed of multiple wards. Each ward has its own staff of nurses. There are, typically, three shifts per day: morning, evening and night, although weekends have some special shifts. The shifts have fixed start and end times that are common to all the wards in the hospital.

2.4.1 Local ward timetabling problem

The nurses of each ward are of several different qualification classes and can be assigned to different tasks in several types of shifts. The requirements of the ward specify how many nurses are needed in each shift, and if nurses with special skills are needed for a certain shift.

Towards the end of each week, the head nurse of each ward constructs the timetable for the nurses of her ward for the following week. This activity consists of the assignment of nurses to shifts according to the requirements of the ward and to the personal constraints and preferences of each nurse in her ward. Each nurse has a list of preferred shifts for the week and a list of forbidden shifts, due to her/his personal constraints.

All nurses have some limit on the total number of hours or shifts that they are allowed to work in one month. In addition, there are several work regulations that impose additional constraints such as: not working more than two consecutive nights or no more than three weekends shifts per month. A special class of nurses may have special constraints or limitations. For example, no more than one junior (student) nurse in one shift, or no more than two morning shifts, per week, for students. In addition there are some global constraints and objectives: distributing night shifts equally among all nurses or distributing weekend shifts equally over a long period. This forms a complex ETP that needs to be solved for each week. Previous research on the basic nurses timetabling problem has been carried using a variety of different techniques (Meisels and Schaefer 2003; Burke et al. 2004). These studies show that the nurses ETPs are, in particular, subject to numerous conflicting constraints and therefore are very difficult to solve.

2.4.2 The nurses transportation problem

Most hospitals management provide some kind of transportation service for the nurses working in the hospital. The transportation service's main goals are picking up the nurses that live in the surrounding towns and returning them home.

Based on the timetables of the wards a transportation plan for all the nurses must be constructed. Since the number of vehicles sent and the number of stops on each line are limited by hospital saving policies, such a transportation plan generates additional constraints between the weekly schedules of the wards in the hospital.

In general, the goal of the transportation plan is to minimize the cost of transportation for the set of transportation requests that are imposed by the departmental assignments. This usually translates into a search to minimize the number of vehicles sent and the distances they must cover. One can model the constraints among timetables of the wards by simple limit constraints on the number of nurses that have to be transported, at the end of their shifts, to several residential locations.

Since the separate timetables for wards are generated locally, the constraints among these timetables, imposed by the limits on transportation vehicles, have to be negotiated among the head nurses of the hospital's wards.

Fig. 1 Reducing the transportation cost is conflict with the quality of timetables

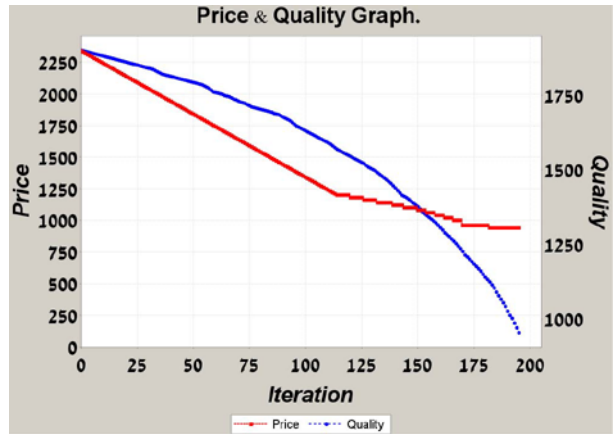







Fig. 2 Soroka’s nurse preferences page

The only work we know of, that proposes a model and presents solving methods for the combination of the basic nurses ETPs and the transportation plan is of Solotorevski et al. (Solotorevsky and Gudes 1996). Solotorevski terms the combination of these two problems *Nurses Timetable and Transportation Problem (NTTP)*. The work of Solotorevsky et al. focuses on finding a legal global timetable, that satisfies all inter-agent constraints. It does so by applying search methods on the distributed constraint satisfaction problem (DisCSP) of timetables and transportation. The focus of the present paper is on the *optimization* problem of the NTTP. This optimization problem is solved by the use of negotiation between the Scheduling Agents.

Almost any simple attempt to lower the cost of transportation is accompanied by timetables of lower quality, for the wards involved. This generates an optimization problem that is currently solved by verbal negotiation between head nurses. The present paper proposes a model and algorithms to automate and replace this verbal negotiation process. The problem of finding a set of timetables for the wards that minimizes the cost of transportation, is a *Distributed Optimization Problem (DCOP)* (Mailler and Lesser 2004).

Table 1 The different employees preference symbols

Symbol	Color	Meaning	Wight
	Dark green	I am specifically asking to work in this specific shift in the next week.	+100
	Light green	I really like to work in this shift.	+10
	Blue	I can work in this shift.	0
	Yellow	I can work in this shift but I really do not like to.	-10
	Red	I refuse to work in this shift.	-10000

2.4.3 Quality of solutions for NTTP

Naturally, one of the goals of the hospital is to minimize the cost of transportation services supplied to its nurses. In simple terms, this target can be achieved easily by filling up each vehicle to its maximal capacity. Such a procedure is in clear conflict with the quality of timetables of each of the wards.

There are many ways to define the quality of a ward's timetable. At the Soroka medical center, for example, the system enables the nurses to define their requests and preferences for working shifts by selecting one of five options. These options are code-colored in the user interface of the system, for each shift, as follows:

One way to quantize the quality of an ward timetable is to count the number of constraint valuation and assign different weight to each constraint type. Much higher weight to hard constraints and lower to soft constraint.

For example, in the Soroka's system there is a fine of -1000000 for any assignment of a nurse to a *red* time slot (i.e. when she is not available), -100 for assignment to *yellow* time slot and 0 for *blue*. In addition, the system assigns a bonus of 100 for an assignment to a *light green* slot and of 200 for *dark green* assignment.

2.5 Distributed constraint satisfaction problems

The nurses timetabling and transportation problem can be modeled as an instance of *Distributed Constraints Satisfaction Problems* (DisCSP). Its *Distributed Constraints Network* (DisCN) has a typical structure, in which the local timetabling problem is quite complex and the global problem includes relatively few constraints. One can think of a DisTTP as a network of scheduling agents, each solving its own local CSP.

Several works consider constraints satisfaction in a distributed form (see Yokoo's book (Yokoo 2001) for an introduction). These works are motivated by the existence of naturally distributed constraints problems, for which it is impossible or undesirable to gather the whole problem knowledge into a single agent and solve it using centralized CSP algorithms. There are several reasons for that. The cost of collecting all information into a single agent could be too high. This includes not only communication costs, but also the cost of translating the problem knowledge into a common format, which could be prohibitive for some applications. Furthermore, gathering all information into a single agent implies that this agent knows every detail about the problem, which could be undesirable for security or privacy reasons (cf. Meseguer and Jimenez 2000). This family of problems is termed *Distributed Constraints Satisfaction Problem* (DisCSP) (Yokoo et al. 1998).

The generic model for distributed timetabling is similar to that of DisCSPs that are solved by a distributed search algorithm (Yokoo et al. 1998; Solotorevsky and Gudes 1996). Agents solve their local assignment problem and interact via messages to make their local schedules compatible with the global constraints (which for the hospital DisETP are constraints of transportation) (Solotorevsky and Gudes 1996).

Each scheduling agent solves the timetabling problem for a single ward. Solving the problem separately for each ward allows the nurses in each ward to set and change their own preferences and constraints independently. In general, each department is composed of different types of employees, has different staffing requirements, and different preferences, this is a typical *Distributed Timetabling Problem* (DisTTP) problem or more specifically a *Distributed Employee Timetabling Problem* (DisETP). Wards can follow different strategies for solving their different problems and can even use a different CSP solver tuned up specifically for their needs.

This distributed agents model is in line with the irreversible trend toward decentralized information sources, a trend that makes the current approach of a global, centralized, search (based on a central computer) less and less adequate.

3 Solving methods for scheduling agents

There are two well known families of algorithms for solving timetabling problems—exhaustive (or complete) search algorithms and stochastic (or local) search algorithms (cf. Meisels and Schaerf 2003). Exhaustive search can go on systematically to find all possible timetables, while stochastic search, typically, produces one solution, reaching a minimum. It has been shown in the past (Schaerf and Meisels 1999) that for real-world ETPs, stochastic search methods outperform complete search in terms of efficiency. Several search algorithms for distributed constraints networks have been proposed in the last decade (cf. Solotorevsky et al. 1996; Yokoo et al. 1998; Bessière et al. 2001; Hamadi and Bessière 1998; Silaghi et al. 2001). However, existing algorithms for solving DisCSPs are all exhaustive. It has been shown that both methods can be useful for solving DisETPs (Meisels and Kaplan-sky 2002).

One possible approach for solving DisETPs uses exhaustive search for the *SAs*. In this approach, the *SAs* perform their exhaustive, systematic, algorithm continuously. Each new local solution they find is sent as a message to the Resources Agent. This method is based on the fact that any global solution will always incorporate one local timetable from each *SA* (i.e. for each ward of the hospital in our example). In order to produce the overall solution, the *CRA* has to search among the set of local solutions, while its domain is constantly enlarged, for one subset that satisfies all the global (inter-agent) constraints. The advantage of the exhaustive search approach is in its high degree of concurrency. In addition, it is guaranteed to find a global solution if such a solution exists. However, for real world problems, complete search can be very slow (Schaerf and Meisels 1999).

A second, more practical method, uses local search for *SAs* to generate their local timetables. Each *SA* can find a local timetable relatively fast, but then it waits for a request, from the resources agent, to improve its solution. This request is the result of the *CRA* attempt to resolve some additional global constraint violations. Upon receiving such a message, an *SA* resumes its local search (i.e. hill climbing) in order to find an improved solution. The advantage of the local search agents approach is that it is likely to converge quickly to some global solution, by using the strategy of improving local solutions. The main drawback of this approach is that the *CRA* may find itself performing cycles in searching for a globally improved solution, since local solutions improve only locally.

3.1 Scheduling agents (SA) solve a dynamic CSP

For the second approach, where each SA uses local search, the level of dynamic CSP goes deeper into the level of the local ETPs. When the CRA gets one solution from each of its SA, the CA computes the current value of the cost function. In the nurses transportation problem, this is the cost of transportation of all the nurses at the start and end of each shift. In the next step, the CRA try to reduce the cost by a request for all or some of its SAs to deliver another solution with some new constraints that the CRA imposes. For example, suppose that in the first set of solutions that each SA delivered to the CRA there is only one nurse a , that needed a transportation to target A and 3 sets of 4 nurses each, to 3 other locations: B , C and D . Furthermore, suppose that there are taxis with capacity of 5 passengers that are available. In such a case, it is easy to see that a reasonable move of the CRA is to ask the SA that makes the schedule for the nurse a , SA_a , to replace the nurse a with any other nurse that lives in location B , C or D . Now SA_a is facing a CSP with constraints that are changing *dynamically*.

In this model each SA must solve a CSP dynamically changing CSP. This is an instance of a known, albeit very little treated, class of CSP problems: the *Dynamic Constraints Satisfaction Problem* (DCSP) (cf. Dechter and Dechter 1988; Verfaillie and Schiex 1994b; Bellicha 1993).

3.2 General iterative restart algorithm—GIRA

All complete search algorithms on CNs fall into the family of intelligent backtracking (cf. Tsang 1993; Prosser 1993). All of these algorithms construct a consistent solution by traversing the search tree of partial assignments to the variables. All complete algorithms keep only partial consistent solutions and if a domain of values of some variable is exhausted, the algorithm backtracks to find different consistent assignments (Tsang 1993). All of these methods, including the best ordering heuristics for variables and values, fail to solve hard enough CNs. It was shown that for large ETPs with dense enough conflicts, only local search methods such as hill climbing solve the problems in practical processing times (Schaerf and Meisels 1999).

Recently, a randomization-of-search method was proposed by Gomes et al. (1998). They propose to randomize complete search by stopping the search procedures that appear to be “stuck” while exploring a part of the space that is far from a solution. Gomes et al. suggest that we better run the search up to a certain *cutoff point* and then restart the search at the root of the tree. This *restart* method is differs from local search. It does not explore the space of complete assignments. Rather, it moves randomly among partial assignments, targeting any path on the search tree that leads to a solution. One set of experiments we report in previous work (Meisels and Kaplansky 2004) demonstrates dramatic reduction of search effort of the GIRA algorithm, compared to complete search on the same backtrack algorithm.

The restart algorithm as proposed in (Gomes et al. 1998) uses a parameter that determines the number of backtracks to perform before restarting the search. This parameter is called the *cutoff parameter* by Gomes et al. (1998). In this paper they discuss the behavior of the restart algorithm and observe that a high value for the cutoff parameter (i.e. many backtrack steps before restarting) leads to poor results on easy problems. Values that are too low for the cutoff parameter may result in failure to solve hard problems. The main idea of our proposed general iterative restart algorithm (GIRA) is to let the algorithm find the suitable value for the cutoff parameter. The GIRA starts the search with an initial value for the cutoff parameter, C . It searches the tree, each time performing up to C backtrack steps and

then *restarts the search*. It also has a second parameter N which determines the number of unsuccessful restarts to perform until the cutoff value is increased.

The following is a pseudo-code for the GIRA.

Algorithm 1 (Iterative restart algorithm)

- 1: Set an initial value S to the "cutoff" parameter C
- 2: **for** $i = 1$ to N **do**
- 3: Run *lgorithmA* for a fixed number of C backtracks.
- 4: **if** A finds a solution or proves it does not exist **then**
- 5: stop
- 6: **else**
- 7: Restart algorithm A from the beginning, using an independent random seed for another C backtracks.
- 8: **end if**
- 9: **end for**
- 10: Multiply C by F .

In the pseudo-code, the variable A stands for any complete backtrack search algorithm. In the present experimental study we use the FC-CBJ algorithm. It is clear that as long as the GIRA Algorithm can not find a solution it will increase the parameter C until it eventually be high enough to change the behavior of GIRA to be similar to complete search algorithm. In other words, for all practical purposes, for large enough C value, GIRA behave as complete search. Therefore we can say that the GIRA is asymptotically a complete search. This is one important advantage of the GIRA.

There are 3 *controlling parameters* to the GIRA algorithm, beside the main choice of the search algorithm A :

C —The *Cutoff parameter*: The number of backtracks the solver executes before restarting the search.

N —The *Number of (idle) Iterations*: How many times to restart before increasing the cut-off parameter.

F —The *Increase Factor parameter*: By what factor to increase the cutoff parameter (after N unsuccessful tries).

The increase of the cutoff parameter C can be any factor $F > 1$. For simplification we choose to increase, after N idle iterations the cutoff parameter C by a factor of 2. Larger values for this parameter will probably deteriorate faster. This iterative process of restarting the search at the root of the tree when C backtracks have been performed and performing this restart N times until the cutoff value is increased, continues until a solution is found. Thus the GIRA introduces systematic randomization into any complete search algorithm, while retaining completeness.

4 A general framework for DisETPs

The work flow in the hospital and the personal nature of the constraints of the nurses in the NTTP are the common reasons why the timetable for each ward is generated locally. The *Shared Resources* of the NTTP are seats at transportation lines for the hospital's nurses. The limits on these shared resources impose inter wards constraints among the wards timetables. Currently, these constraints must be verbally negotiated among ward head nurses. The approach proposed in this paper uses a multi stage inter-agent negotiation procedure (see

Fig. 3 A simple model for small scale DisETPs. The Central Resources Agent (*CRA*) adds new constraints to the scheduling agents (*SAs*). The *SAs* solve the modified problem and send back a new timetable

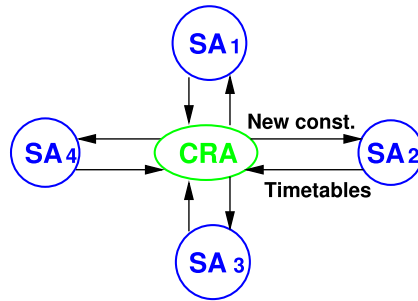
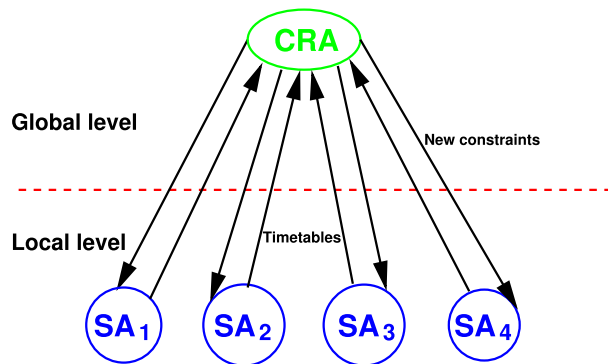


Fig. 4 Two levels of the DisETP model: global and local



Sect. 7). Several alternative architectures to support such inter-agent negotiation were investigated.

4.1 Distributed resource management

A simple model for DisETPs is shown in Fig. 3. In this architecture, an additional agent that manages the shared resources is introduced. We term this agent the *Central Resource Agent* (*CRA*). The *CRA* deals with all the constraints between the *SAs*. All agents communicate by sending and receiving messages from the *CRA*. *SAs* send their request for using the common resource to the *CRA*. The *CRA* checks for conflicts with global constraints and decides which agents have to be requested to change their local timetables.

Based on this model, there are two different levels (see Fig. 4) that need to be dealt with in order to solve a distributed timetabling problem:

The global level: The solution methodology for the *Central Resources Agent* searches for a combination of different *SAs* solutions that can satisfy the global constraints of the problem.

In many cases the search is also for an optimal global solution. A detailed description of solving methods for the *CRA* are given in Sect. 4.4.

The local level: Search algorithms for each *SA* that can solve a local ETP where constraints are added or removed dynamically. This level is discussed in Sect. 3.1.

4.2 Simple search control

The naive way to search for a global consisted solution is to let the *SAs* to search continuously for alternative solutions to their local ETPs. In this protocol the Central Resources

Agent (*CRA*) has no control over the search of each of the *Scheduling Agents*. The *CRA* manages a pool of alternative local solutions and searches the pool to find a combination of local solutions that are compatible with each other. The behavior of this simple search control was investigated experimentally. Initially, the *CRA* must wait until all *SAs* return with at least one solution. Thereafter, the *CRA* starts its own search for a legal combination of *SAs*' solutions satisfying the global constraints of the problem. The initial solution of each *SA* is based on the transportation plan of the previous week. In this simple protocol the *SAs* are constantly busy, searching for alternative solutions.

Preliminary results show that in this simple protocol the search space of the *CRA* can grow very fast and usually, quite soon, it becomes prohibitively large. For example: 10 departments, each with 100 alternative timetables result in 10^{20} nodes on the *CRA*'s search tree. To deal with this size the *CRA* has to use more efficient search method for finding an optimal solution in a reasonable time frame.

In the beginning, when the solution pool is small, the *CRA* can start with complete search. When too many alternative timetables have arrived and the pool becomes too large, the *CRA* moves to local search using a *Random Hill climbing* (RHC) algorithm. Eventually, the *CRA* performs *Tabu search* that is based on scanning the neighborhood around the current solution (Glover and Laguna 1997; Burke et al. 1998).

For difficult ETPs the *SAs* generate only few alternative schedules per time unit. For such problems the *CRA* needs to use a heuristic that will enhance the chance of encountering lower transportation cost for the next *SA* solution. The *CRA* requests the *SAs* to generate timetables with needed features in order to avoid violating the capacity constraints of the transportation lines. The *TA* sends messages that add a constraint to a selected set of *SAs*, to limit the number of nurses for some line. In addition, new constraints are needed to prevent the *SAs* from adding a nurse to a line with full cabs. Another version of a heuristic is to add new constraints that forces *SAs* to reduce nurses in lines with an almost empty cab.

4.3 Experimental evaluation

In order to understand the basic behavior of the simple architecture model, a set of experiments were conducted. The experiments use a set of problems for timetabling that are similar to the ETP of a typical ward in Soroka hospital. In order to explore a wide range of the problem space, many instances of this real world ETP were generated with varying amounts of binary constraints, by adding or removing conflicts among shifts. Adding conflicts among shifts creates a random sample of problems with the same underlying structure (i.e., shifts; tasks; employees; limits) and different domains of values. This has the additional experimental value of a large random ensemble of problems.

In the first set of experiments that is presented in Fig. 5 the effect of using local search techniques for the *CRA* search is compared with the best complete search algorithm: FC-CBJ (Prosser 1993). Each point on the graph of Fig. 1 represents the average of a set of 100 problem instances. The stopping criterion for all unsuccessful runs is based on the total number of assignment attempts by the algorithm and that limit was set to 500,000 assignments. Using the number of iterations performed as the measurement unit for the search effort, we compare the efforts needed to solve the same set of problems by a pure FC-CBJ algorithm and by the hybrid local search algorithm. It is easy to see that local search finds a solution in about 1/3 of the time it takes FC-CBJ. The solutions found by local search are also about 10% better.

The above result leads to the understanding that in this type of ETPs it is important for the *CRA* to control its *SAs* in such a way that the *CRA* can accumulate a promising set of

Fig. 5 Comparing complete search to local search: at some point, too many alternative solutions make the complete search ineffective

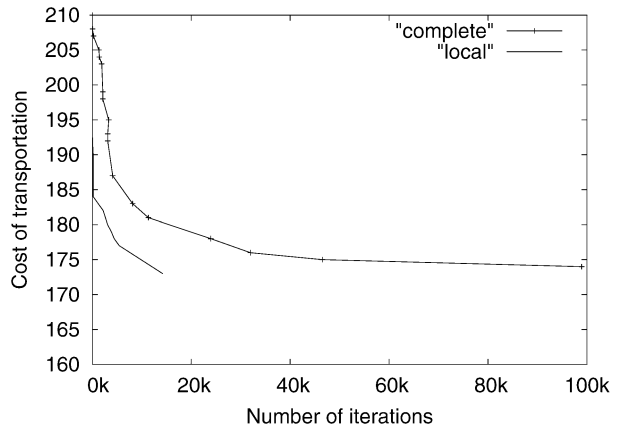
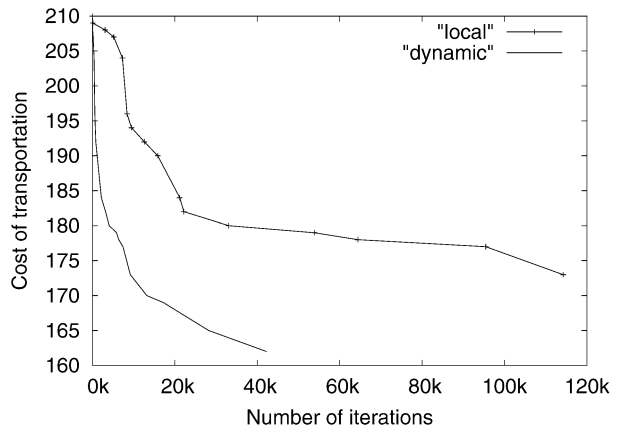


Fig. 6 Using sophisticated interaction between the *CRA* and its *SAs* enables the search to find a better solution faster



alternative local solutions in a short time. At the end of each time slice the *CRA* examines the best transportation plan and selects one target to improve. In this protocol the *CRA* uses two types of control messages:

1. No transportation line available for a given shift and a given destination.
2. Additional seats are available for given shift to given destination.

The next step of investigation is to check the effectiveness of intelligent interaction between the *TA* and its *SAs*. In other words, the importance of a good strategy for the *CRA* search method. The results shown in Fig. 6 show that the *CRA* agent must search smartly and coordinate the *SAs* in order to find better combinations of *SAs*' timetables. In the algorithm marked as “dynamic”, the *CRA* dynamically adds and removes constraints to the problems of the *SAs*. The best solution with this method has a cost that is lower by 10% than a simple local search. From these experiments we can conclude that the *CRA* needs specific heuristics for directing *SAs* to improve their timetables.

4.4 Search algorithm for the resource agent

The main difficulty of the search problem of the Central Resources Agent (*CRA*) stems from the fact that the values for assignment in the *CRA* constraint network are acquired

incrementally from the scheduling agents. This transforms the *CRA* constraints satisfaction problem into a special class of Dynamic CSPs (DCSP) (Verfaillie and Schiex 1994a).

One of the basic operations of this scan is a request for an alternative timetable from some SA. The time that a SA needs for computing the next solution varies widely. One possible solution for this difficulty is that the *CRA* will manage a dynamic buffer of alternative timetables. Another approach for the *CRA* is to request multiple alternative solutions simultaneously from a selected group of SAs at each step of the local search. This forms a set of intelligent improvement moves.

In the experiments of the present paper, the *CRA* searches for an optimal global solution. In such a paradigm the basic operation for the *CRA* is to request an alternative solution from one of its SAs. One way to guide the SA is by dynamically adding new constraints to the SA's local problem. In our problem, the *CRA* can find out, for example, that on the night shift of Wednesday, there are 12 nurses from same given town. This is one nurses over the limit of 11. In order to satisfy this global constraint, the *CRA* can select the SA with the largest number of requests for seats on this line and send it a message, asking this SA to search for an alternative local solution. This new local solution has to satisfy an additional constraint: "No more than 4 nurses living in that town, for Wednesday night shift." Each new SA solution is an additional new value to one variable of the *CRA* constraints network.

When one tries to solve a DisETP, one needs to solve ETP problems that are *dynamically* changing. The basic idea, here, is that each SA starts with its original ETP problem. Whenever any of the SAs finds a solution it sends this solution to the *resources Agent*. When the *CRA* gets at least one solution from each of the SAs, it starts a search for a solution to the *Global CSP problem* (satisfying the *Global resource constraints*). Each variable of the GCSP represent one SA. The domain of each variable is the set of solutions for the local problem of its SA. At the start of the search, the domains include only the first solution found by the SAs. Then the domain of each variable of the *CRA*, is dynamically increased as the SAs find more solutions that complies with all of its local constraints. In this model the *CRA* solves a CSP whose variable domains are dynamically changing.

It can be said that the *CRA* operates in an environment that is very similar to an open-world environment of the Internet. Many real world problems that used to be solved by traditional CSP techniques, can be solved more effectively in a distributed setting. For example, for the problem of computer configuration, it is now possible, to locate through the Internet, on-line, during the search for a solution, additional suppliers of some needed parts. The term Open Constraint Satisfaction Problem (OCSP) was first used for such a search environment by Faltings and Macho-Gonzalez (2002).

Most successful CSP solving methods, in particular constraint propagation, are based on the closed-world assumption where the domains of the variables are completely known and are fixed during the search process. For an Open CSP setting, this assumption no longer holds (cf. Nguyen and Deville 1998; Bessière 1991, 1992; Bellicha 1993). This change requires an adaptation of the search algorithm.

Open constraint satisfaction problems bear some resemblance to interactive constraint satisfaction (ICSP) introduced by Lamma et al. in 1999 (Lamma et al. 1999). In the ICSP model, domain values are acquired during the solution process only when necessary, and inserted into the variable domains. In this methodology the forward checking (FC) algorithm can be modified so that when domains become empty, it launches a specific request for additional values that can satisfy the constraints on that variable. Similarly to our *CRA* search problem, the ICSP acquisition process of new domain values from external agents needs to be guided in order to acquire only new values that satisfy a specific constraint. In the ICSP model the acquisition of new values is an immediate interactive operation and it typically

gathers more values than necessary. In the DisETP model, a request from the *CRA* for a new solution from a *SA* is usually a very expensive operation and the focus is on minimizing the information gathering activity.

5 Solving DisETPs by negotiation

Experimental study of the architecture described above shows it to be very effective for small scale ETPs with relatively few scheduling agents (i.e. wards). This architecture can also find a cost effective solution in a reasonable time frame. However, when applying this simple architecture to real world ETPs, where the number of involved agents is large, the number of inter agent constraints grows exponentially and the problem becomes too large to be solved by one *CRA*. The size and the complexity of the *CRA*'s problem make it unrealistic for one *CRA* to find a solution in a reasonable time frame, let alone finding an optimal solution.

Several alternative architectures that can mitigate the difficulty of the problem of the *CRA* were investigated. The first protocol gives the *CRA* almost full control of the distributed search. Next, we add concurrency to the architecture. The third protocol is more sophisticated. This protocol adds several agents to the architecture, where each such agent is responsible for managing one of the shared resources. For the NTTP, this means generating one additional agent for managing each of the shared transportation lines.

5.1 Synchronized protocol for the resource agent

Previous work on ETPs (Meisels and Kaplansky 2004) had shown that there are families of ETPs for which the time needed to find the first solution varies widely between instances of the same family. On the other hand, there are ETP families that are very easy to solve. For ETPs of the harder family, it is intuitively useful to use a search control that has tighter control than for the easier ETP families. The following *Synchronized Protocol* gives the Central Resource Agent full control of the distributed search.

The synchronized protocol consist of 8 steps:

Step 1—Calculating initial local solution:

Each Scheduling Agent (*SA*) computes the best local solution when it is free to place its employees on any time slot regardless of the availability of any transportation line it needs. Based on this *initial local solution* the *SAs* send the *Initial Request For Seats (IRfS)* to the Central Resource Agent (*CRA*). The *IRfS* is accompanied by an index that represent the quality of the local solution.

Step 2—Calculating initial global solution:

After the *CRA* received the *Initial Requests For Seats* from all the *SAs*, the *CRA* sums them up into the *Global Seats table (GST)*. The *GST* has one row for each shift and one column for each transportation line. Any given entry in the *GST* specified the demand for seats for a given shift and a given destination.

Step 3—Detecting global hard conflicts:

The hospital enforces hard constraints on the use of transportation vehicles. In this protocol, two global constraints were implemented:

1. No fewer then 3 passengers on a vehicle.
2. No more than 11 passengers per one transportation.

In this step the *CRA* consider all the entries in the *GST* that have less than 3 passengers and tries to nullify them (i.e. to cancel these transportation line). Next, the *CRA* examines the

Soroka University Medical Center
האוניברסיטה סורוקה

Controller Console

Initial Sol. Current Sol. Configure Strategy Graphs

Step Iterate Run Stop Log Full Clear Exit

Initializing Line Managers
Received initial solution from department 1
Received initial solution from department 2
Received initial solution from department 3
Received initial solution from department 4
Received all initial solutions

Initial Solution:
01: 1 3 4 14 9 12 0 0 6 14 4 12 7
02: 5 4 1 0 9 0 0 5 0 9 13
03: 1 8 8 11 1 8 13 0 14 10 3 7
04: 4 12 4 0 12 4 4 14 7 10 11 9
05: 14 0 6 11 13 13 7 11 9 11 9 3
06: 3 4 7 2 3 2 8 3 1 9 13 1
07: 6 14 0 0 3 0 4 12 13 12 4 3
08: 8 9 12 10 14 6 10 2 0 2 9 9
09: 6 6 10 10 9 5 12 12 3 6 8 4
10: 5 14 12 9 6 1 13 8 1 1 4 13
11: 8 4 9 14 1 9 4 2 0 1 1 7
12: 11 8 10 4 10 5 2 3 10 13 14 4
13: 3 5 7 10 12 12 13 5 14 2 4 3
14: 9 1 7 14 2 8 10 4 4 0 11 7
15: 1 4 5 11 2 13 1 0 6 8 2 3
16: 7 2 6 9 14 3 4 6 0 12 6 2
17: 9 13 4 12 10 7 4 8 13 11 0 1
18: 1 2 5 1 8 14 0 11 7 8 14 10
19: 12 1 11 13 10 11 8 11 14 3 11 11
20: 5 9 10 12 6 8 10 5 7 4 14 11
21: 6 9 9 9 13 6 14 14 0 5 1 11

The following line masters were found :
Line #0
Master Department: 1
Slaves:

Current Price 3900 Quality 215

Fig. 7 The monitor of the Central Resource Agent CRA at Soroka. The log screen shows the initial Global Seats table (GST)

column of such an entry for *alternative Time Slots (ATS)*, that have entries with number of passenger between 2 to 9. An *ATS* should be the same day shift (morning, evening or night) as the entry it tries to nullify. If the *CRA* can find at least one *ATS*, then the *CRA* sends a *Request for Change (RfC)* to any *SA* that requested seats for this entry. The *RfC* informs the *SA* that there is no transportation line available for this shift. The *RfC* message is accompanied by a list of suggested *ATSs* and *Time Limit for an Answer (TLA)*.

Step 4—Resolving global hard constraints:

When a *SA* get a *Request for Change (RfC)* it adds it as a new constraint to its local ETP and runs its solver up to the given *TLA*. When the *TLA* time is up, the *SA* selects the best alternative solution it had succeeded to find. If the new solution's quality is in the ward's permitted range, the *SA* sends the *CRA* an answer with the new *Table of Demands for Seats*. Otherwise, the *SA* sends a message that gives up the seats for this entry.¹

The *CRA* waits for answer messages from all the *SAs*. These answers can include a new *Table of Demands for Seats* or a *Give up the seat* message. With these answers, the *CRA* can construct a conflicts free global solution.

Step 5—Global optimization loop:

In this step, the *CRA* starts a loop of iterations consisting of 3 steps (step 4–6). The *CRA*'s target is to improve the quality of the global solution. The *CRA* search the *GST* for candidate entries that it estimates to have a higher probability to be nullify. For each such candidate entry in the *GST* the *CRA* sends a *Suggestion for Change (SfC)* message to all *SAs* that are invoked in this *GST* entry. The *SfC* message is accompanied by a list of suggested *ATSs* and *Time Limit for an Answer (TLA)*.

Step 6—Local search:

The *Suggestion for Change (SfC)* message invokes a new local search at the *SA*. The *SA* can responses with *Approve Message* or a *Decline Message* to the *CRA*. An *Approve Message* is accompanied with a new *Table of Demands for Seats* and a new *Solution Quality Index*.

Step 7—A global improvement:

The *CRA* waits for answer messages from all the *SAs* that are involved in the iteration. If all involved *SAs* return an *Approve Message* a new global solution is approved. In this case the *CRA* update the *Global Solution Quality Index (GSQI)* and sends a *keep the new solution* to all involved *SAs*.

Otherwise, when even one *SAs* refuses to change its timetable the iteration is considered a failure and the corresponding *GTS* entry is pushed into a *Tabu List*.

As long as the *GSQI* is still in the hospital permitted range The *CRA* repeats the *Global optimization Loop*, Step 5 to 7.

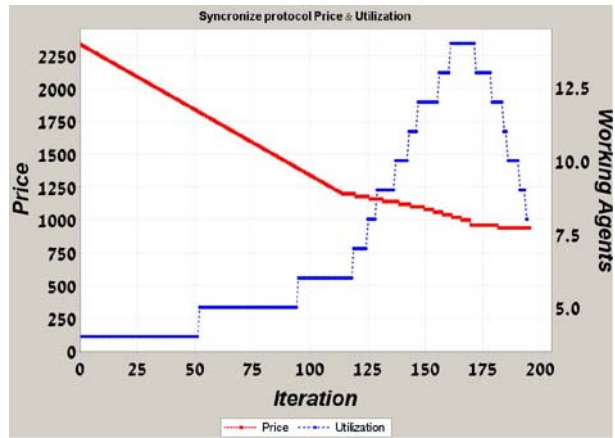
Step 8—Stopping criterion:

Anytime a *Global Optimization Iteration* fails, the corresponding *GST* entry is pushed into the *Tabu List*. An entry can be taken out from the *Tabu List* a limited number of times. When the *CRA* tries all potential entries in the *GST* for the limited number of times, the global search is stopped and the current solution is return.

The *Global optimization loop* in the above protocol is actually run for two rounds. In the first round, as described above, the target is to nullify transportation lines with a small number of passengers. In the second round the *CRA* tries to reduce the number of passengers that are above the vehicles capacity. This round runs similarly to the first round, except for two changes:

¹In Soroka hospital, in this case, the nurse can agree to use other transportation means. Alternatively, the ward's nurse head can request the hospital management to use a transportation line with less than 3 passengers.

Fig. 8 Solving with the *Synchronized Protocol*, the utilization of the *SAs* is about 10%



1. The new constraint for the *SA* is:
The number of seat on a given line must be less than the number of seats request by the *SA*.
2. In Step 7, it is enough that only one *SA* returns with *Approve Message* for this improvement loop to be considered a success. It is assumed that reducing the number of superfluous passengers by only one passenger, improves the global quality of the solution. This protocol enables the *CRA* to try an entry for several more times, and there is a chance to furthermore reduction in the number of superfluous passengers.²

5.2 Improving concurrency for the *SAs*

Experiments conducted with the *Synchronized Protocol* on sets of real world ETPs of Soroka show that during the life time of the negotiation process most of the *SAs* are idle. In fact, only about 15% of the *SAs* are busy performing either local search or negotiation with the *CRA*. In order to make the DisETP search more efficient, concurrency was introduced into the *Synchronized Protocol*.

The activities in the *Concurrent Search Protocol* are the same as in the *Synchronized Protocol* except for that there being several *Global optimization loops* that are running in parallel.

The changes that enable this concurrency are the following:

Step 5—Global optimization loop:

In order to enable concurrency, the *CRA* maintains a queue of n potential candidates *GST*'s entries as targets to nullify.

Next, the *CRA* dequeues the first candidate entry and starts a new *Global optimization loop* to negotiate the change of seats demands corresponding to this entry. The *CRA* keep a list of *SA* that are currently busy in local search for a new alternative local timetable.

Next, the *CRA* dequeues the next candidate entry and checks whether any of the *SAs* involved are listed in the *Currently Busy Agents List*. In such a case it returns this entry to the tail of the queue. Otherwise, the *CRA* launches a new *Global optimization loop*.

²This protocol feature is based on Soroka's policy that allow to split a transportation line to two for which demand exceed its capacity.

Fig. 9 For the *Concurrent Search* protocol, the utilization of the *SAs* time is about 50%

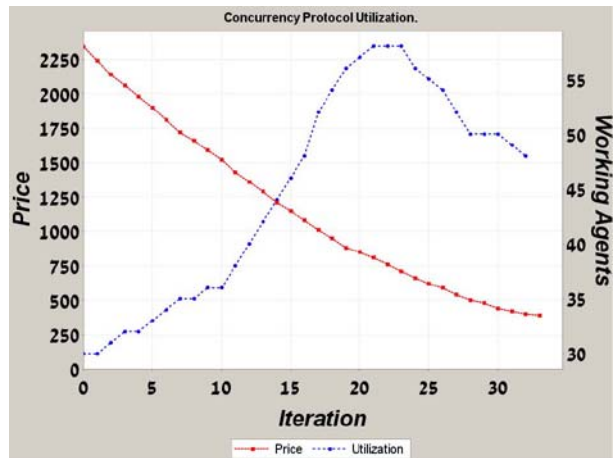
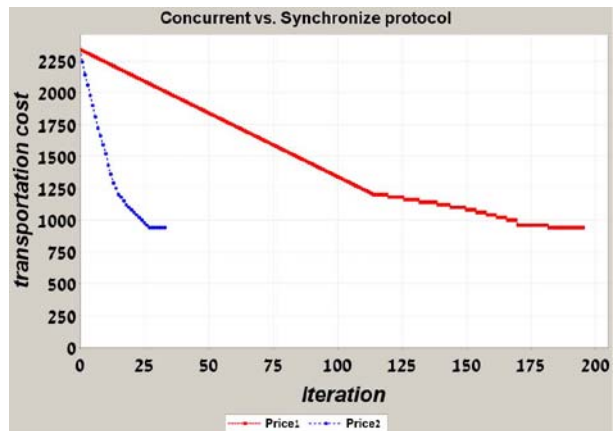


Fig. 10 Adding concurrency to the *Synchronized protocol* produce same results faster



The *CRA* tries to launch a *Global optimization loop* for all the original n potential candidates entries.

Next, move on to *Step 7* to take care of the incoming messages.

Step 7—A global improvement:

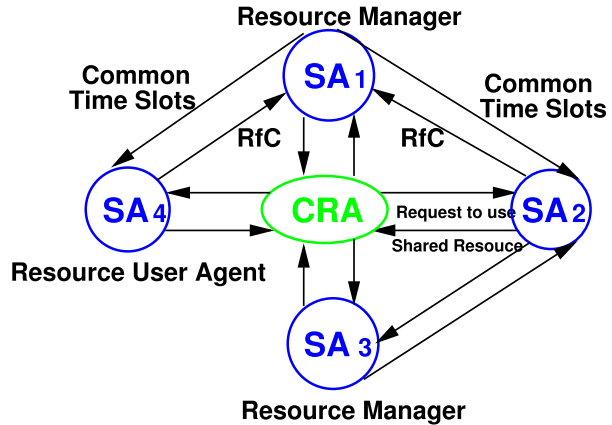
Rather than waiting for *all* answer messages, in this protocol, the *CRA* examines the buffer of the answer messages from the *SAs*. For any answer it reads, the *CRA* remove the answering *SAs* from the *Currently Busy Agents List* and returns to *Step 5* (scan the candidate entry queue for any potential candidates that now can be lunched). When the incoming messages buffer is empty, *SAs* perform the same activities as in *Step 7* of the *Synchronized protocol*.

5.3 Solving by resource managers

In a more sophisticated architecture, parts of the problem of the *CRA* can be delegated to other *SAs*. In this architecture, one *SA*, that uses a specific shared resource, is selected to be responsible for this resource. This agent is called a *Resource Manager Agent* (RMA). The other agents, that use this resource, are termed *Resource User Agents* (RUAs).

For this architecture to be effective, a well defined protocol is needed. In order to agree on a common time slot for all agents that need to use a given resource, the protocol enables

Fig. 11 Parts of the *CRA*'s problem is delegated to other *SAs*. The *SAs* conduct negotiation for global timetable. Next the *SAs* send the *CRA* request to use the shared resource



each *RMA* to conduct negotiations with all the other agents that use the same resource. Once an agreement is reached between all the agents involved, the *RMA* submits a request to the *CRA* to use this resource in the agreed time slot. The *CRA* can accept or reject the *RMA*'s requests. This involves an additional protocol that can support negotiation between the *CRA* and the *RMA*s.

The *Resource Manager Protocol* is a modification of the *Synchronized Protocol*. Here, the task of the *CRA* is reduced to start the protocol and collect the final solution.

At the end of Step 2 of the *Resource Manager Protocol* the *CRA* finds for each line, the *SA* that requested the largest number of seats for the whole week and sends this *SA* a message. When the *SA* gets this message, it launches another software agent: the *Resource Manager Agent* (*RMA*) of this line. The *RMA* follows the same procedure as the original *CRA* for the single line it manages. At the end of the negotiation between the *RMA* to the *SAs*, the *RAM* sends a *Final Request For Seats (FRfS)* to the Central Resource Agent.

The change in the activity of the *SAs* from the synchronized case to the present architecture are minor. While the formers two protocols ensure that each *SA* deals with one task at a time, in present protocol the *SAs* negotiate with several *RMA*s concurrently. The concurrency of the negotiation between *SAs* and several *RMA*s generate the need to keep track of different alternative solutions the *SA* had proposed to the *RMA*s.

When the *SA* receives approval for a proposal solution, it uses solutions that do not conflict with other proposals. A simple way to manage this situation, is to decline any request for change that conflicts with previous proposals. This policy proves to be quite successful as can be seen in Fig. 12. The figure presents a comparison of performance of the *Resource Manager Protocol* and of the results of the *Concurrent Search Protocol*. The improved converge of the *Resource Manager Protocol* can be explained by the much higher concurrency of the *SAs*. The tight loop of the *RMA* keep requesting the same changes from the *SAs*. This generates repeated search by the *SAs*, which in turn gives a better solution and performance. An improved version is presented in Sect. 7.

6 The DisETP system at SOROKA

The Soroka Medical Center is Israel's second largest medical center, with more than 1,200 beds and a staff of nearly 4000. Soroka is located in Beer-Sheva and it is the only hospital serving the entire population of southern Israel, the Negev region, that constitutes 60% of

the country's land mass, close to one million people. Soroka also serves as the nucleus of the Ben Gurion University Faculty of Health Sciences. Last year, 188,000 patients, including 38,000 children, visited Soroka's emergency rooms, making it the largest emergency facility in Israel. Soroka has 54 wards that handle about 12,000 births, 90,000 hospitalizations and 22,000 operations each year.

Soroka has staff of about 600 physicians and more than 1200 nurses. The nurses in Soroka come both from the city where the hospital is located and from several surrounding towns which are up to 60 km away from Beer-Sheva.

The hospital management provides transportation services for picking up the nurses that live in the surrounding towns from their homes to the hospital and returning them home after their shift. The unit responsible for the transportation service is called the transportation center (TC).

The TC serves the nurses that work evening and night shifts. (The assumption is that the nurses that work on regular day shifts can use the public transportation services.) On regular weekdays the TC provides 4 sets of lines:

1. *Evening pick up*: Starting at about 14:30, to pick up the nurses from home to the evening shift.
2. *Night pick up*: Starting at about 21:30 to pick up the nurses from home to the night shift.
3. *Evening distribution*: Return nurses home at the end of the evening shift—leaving the hospital at about 23:30.
4. *Night distribution*: Return nurses home at the end of the night shift—leaving the hospital at about 07:30 the next day.

On weekends and holidays the TC provides transportation for all 3 shifts.

The transportation service maintains 12 lines. Each line serving several towns and using taxis or mini-buses that take up to 11 passengers. The hospital hires the vehicles according to weekly demands. The hospital may hire, for a given line more than one vehicle at a certain time and none at another time. A taxi may go only to towns 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/picks passengers. The hospital pays the transportation company a basic fee per vehicle, per line, ignoring the number of passengers. There is additional fee according to the number of towns the taxi actually entered.

For example, the basic cost of one vehicle on the line to Arad is 150 NIS. This pay includes 2 town entries. There is an additional cost of 50 NIs for each additional entry. In this example, if there are 10 nurses asking to use this line, but they live in 4 different towns the cost is 250 ($150 + 50 \times 2$). However, there is a limit on the number of stops for one line. Too many entries may take too long.

The current status of the work flow at Soroka is, first, to solve the assignment problem of the wards incorporating regulations and nurses' objectives and constraints. Next, the schedules of the wards are checked against the constraints of transportation. If found compatible, nurses are assigned to the lines trying to achieve a minimal cost. Otherwise, the wards are asked to make some changes in their timetables. This cycle can be repeated multiple times. The number of nurses served by each of the lines in Soroka varies greatly.

The basic constraint imposed by the hospital is that no vehicle can be hired if there are less than 3 passengers for that line. We term this constraint *Minimum Passengers constraint* (MinPas). When the number of nurses served by a line is relatively large, the MinPas constraint for that line will rarely rule out a feasible set of timetables for the wards. In contrast, there are lines that serve only about 10 nurses. For such lines the MinPas constraint frequently causes a conflict with the timetables proposed by the wards.

A project to implement a distributed employee timetabling (DisETP) system for SOROKA medical center is an ongoing work. The first module that has been implemented for each ward serves the nurses weekly timetable. This module enables Soroka to collect real relevant data, mainly:

1. The weekly timetable of each ward.
2. The nurses home location, personal constraints and preferences.
3. The cost and availability of the transportation lines.

This module realizes our concept of a Scheduling Agent.

The first version of the *Transportation Agent*, which assigns nurses to transportation lines with the constraints of the vehicles capacity, is now being tested. It interacts with the *SAs* of the wards and uses the negotiation protocol in order to get a feasible transportation plan for the week. The distributed system implements the negotiation protocol among *SAs* and the TA. It implements all needed mechanisms for dealing with *Requests for change* and the protocol of stage II for an optimal hospital-wide consistent schedule.

6.1 Negotiation performance at Soroka

In order to understand the basic behavior of the proposed negotiation model, we have conducted a set of experiments. The experiments use a set of problems for timetabling of several wards in Soroka hospital. Each ward schedules its nurses independently. Transportation lines are grouped into destinations. The negotiation protocol for the first and second stage (see Sect. 2) force the *SAs* to coordinate their timetables. *SAs* are not aware of the inter-ward transportation constraints. These are checked and computed by the *Transportation Agent*.

In standard ward timetable problem that is used in the experiments there are 21 shifts per week. All weekly timetables are similar—36 nurses have to be assigned to 3 tasks for a total of 112 assignments over 7 days. There are 12 lines of transportation. Cabs have a capacity of 1–7 passengers. Bus capacities are fixed to 11. There are different costs for lines for different shifts.

7 General architecture for negotiation

In our model, inter agent negotiation consists of three main stages. The first stage (steps 1–3 below) focuses on search for a local solution for the timetabling problem of each ward. In stage 2 (steps 4–8), the target is a initial global timetable that is globally acceptable with respect to the limitation on the common resources. In the third stage the *SAs* negotiate in order to improve the quality of their local solution. For the NTTP problem this takes the form of negotiating for seats on the transportation lines, in time slots that are preferred by the nurses of each ward.

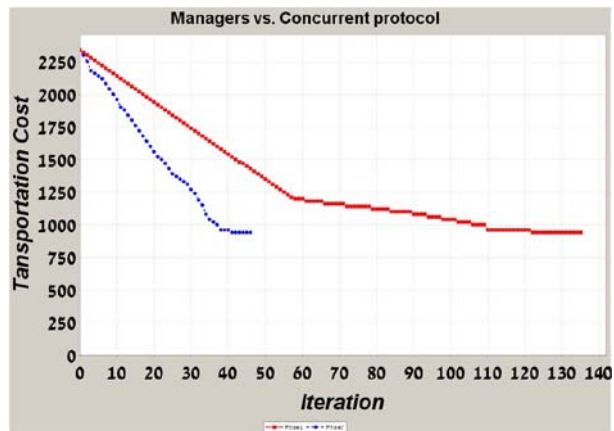
Stage I: Setting the negotiation infrastructure

In this stage the *SAs* prepare the infrastructure for an effective inter agent negotiation.

Step 1—Self assessment:

Each agent estimates the rate of finding alternative solutions for its local problem, expressed by the *Average Computation Cost (ACC)* per solution found. ACC is measured by the number of *constraints checks* per solution found. At this stage, the specific time slots of the shared resources are still unknown. In order to make this local solution more realistic, with respect to the availability of the shared resources, the agents can take one of the following intelligent guesses:

Fig. 12 Concurrent protocol vs. resource managers architecture



Last week: Use last week's time slots. This is the option taken today when the timetables are done manually.

Random: Use random time slots with some known guidelines.

Manually: Let the user set these slots manually.

In the second part of the self assessment step, the agent calculates its *Agent's Best Quality Index (ABestI)*. This is an indicator for the quality of the best solution the agent can find when it is free to place its employees on any time slot regardless of the availability of any shared resource it needs. In the NTTTP this means the freedom to assign the nurses to any shift on any day. *ABestI* represents an absolute upper limit of the quality of any solution for the agent's local problem.

Step 2—Setting the inter-agent relations:

Preliminary experiments had shown that the complexity of the local problem of each agent is a significant factor for choosing an agent to be the manager of a specific resource.

Each agent sends its ACC, calculated in step 1, to all its constraining agents.

Agents agree on the direction of the constraint between them in such a way that the most complex (slowest) agent is the *Resource Manager Agent* and the other agents are the *Resource User Agents*. To avoid cycles, a directed spanning tree over the whole network of SAs is calculated by a version of a distributed DFS algorithm. In this way, each non-binary constraint is translated into a set of binary constraints.

Step 3—Calculate best local solution:

In order to achieve initial global solution that it is compatible with respect to the time slots of the shared resources, each agent that uses a ShRes received the available time slots from all its resource manager agents. The relationship of *manager–user* are established by the previous step.

Time slot conflicts at the receiving agent can occur when a Resource Manager Agent sets the time slot of the ShRes differently than the time slot needed by the receiving agent. Such conflicts are resolved in the next stage. The Resource Manager Agent approves the request of the most complex agent (as established in step 1) and sends new constraints to the other *Resource Manager Agents*.

Next, each agent computes its best local solution with the assignments that need shared resources set to fixed given time slots. Finally, the quality index of the best solution found is termed *Agent Minimal Quality Index (AMinQI)*.

Stage II: Constructing a globally feasible solution

In this stage, a conflict free global solution is achieved by a negotiation protocol among the *SAs* and the *RMA*s. Each resource manager propagates the time-slots of its solution to the agents using the resources it manages. User agents issue requests for change in order to be compatible with other agents. The right to issue a request for change is transferred from one agent to the next agent by a procedure similar to a *token ring*. Requests for change can be approved or refused by the resource managers.

This stage is iterative. In each iteration the *token* is moved to the agent with the probability to achieve the highest quality gain when its request will be approved. The algorithm to select the next agent to receive the *token* is similar to the know, distributed algorithm for *Leader Election*.

Step 4—Initial suggestion: Every resource manager sends the shifts of the line it manages to the agents that use these resources, on its outgoing links.

In response, each *SA* sends its *Agent Cost Range* $ARangeC = AMaxC - AMinC$ to the agents that manage its set of lines.

Step 5—Normalizing Agents' Costs: To enable the resource manager to resolve conflicts between agents sharing the same resource, the manager generates a common scale by normalizing the complexity of all agents using this resource. For each shared resource, the resource manager agent computes a set of *resource user weights*, that scales the *ARangeC* of each agent that uses this resource to 1.

Step 6—Requests for Change: Each agent tries to improve its local timetable by changing the time of one of its shared line and re-solving its local problem. When it finds a better solution, it sends a *Request for Change* (RfC) to the *Resource Manager Agent*. The RfC message is accompanied by an *Expected Quality Gain* (EG) and a list of suggested *Alternative Time Slots* (ATS). This agent is termed the *Initiator Improvement Agent* (IIA).

Step 7—Approve a change: When a *Resource Manager Agent* receives a *Request for Change*, it searches for the best solution using any one of the slots of the ATS list accompanying the RfC. If the *Manager Change Cost* is lower than the *Expected Gain* an *Approve Change Procedure* is started:

1. The resource manager sends a *Change Offer* to all agents using this resource.
2. All *Line Users Change Costs* are collected.
3. If the sum of all *Change Costs* is lower than the *Expected Gain* then this *Request for Change* is approved. Otherwise, this *Request for Change* is refused. This conflict can be solved in one of the following:
 - In one of the following iterations.
 - The next step—the bid.

Step 8—Next Iteration: In this step, the agent that holds the *Change Token* initiates a distributed search algorithm to select the next agent to get the token. Once it is agreed, it passes the token on, and another iteration of request for change is started.

Stage III: Improving quality of local solutions

In this stage, the *SAs* try to improve the quality of their local solutions.

Step 9—The bid: In the case that a *RMA* refuses to approve one of the user agent's request for change, the *Initiator Improvement Agent* attempts to buy its right to change the time slot of a shared resource from the *RMA* using its *bid tokens*. The *RMA*, tries, in its turn, to use these tokens for buying this RfC from all the other agents that use this resource. If the bid succeeds, the RfC is approved.

Step 10—Negotiations with the Central Resources Agent In general, the shared resources are not infinite, but rather, have some upper limit. In the NTTP case, there is a limit on the number of seats in each line. In our case study the limit is 11 passengers per vehicle. When a hospital wide stable global timetable is reached, each agent sends a *Seats Request* for each line it manages to the *CRA*.

If demand for line exceeds the limit for some specific line, the Transportation Agent send a *Seat Refusal Message* to all agents that request seats for this line accompanied by *ATS*. For example, offering the nurses to work in a different night shift, where there are free seats available.

The *SAs* try to change the time slot of nurses that are assigned to this *Refusal Message Shift*. If the *RMA* gets enough response messages agreeing to change—the problem is solved.

Otherwise, the *RMA* initializes an auction for seats for this line. A separate auction process is managed by the *CRA* in order to resolve each case of *Not enough seats*.

8 Conclusions

The *Distributed Employee Timetabling Problem* (DisETP) has been presented in detail. A distributed timetabling problem is composed of *Scheduling Agents* (*SAs*) that solve different parts of the organization's timetabling problem and *Resource Manager Agents* (*RMA*s) that manage the resources that are used by multiple *SAs*. Scheduling agents are connected by few global constraints, such as limits on seats available on transportation lines for nurses. When the *CRA* performs an exhaustive search for a global solution for the DisETP, the problem can become prohibitively large. This is because each node on the search tree of the *CRA* is a complete solution of the *SAs'* timetables.

Our model and approaches were implemented for a real world distributed employee timetabling problem (DisETP) involving multiple wards of a large hospital. Each of the *SAs* solves an ETP with more than one hundred needed assignments. The simplest protocol for the *CRA* was tested to show intuitive results.

When the DisETP is composed of a realistic number of *SAs* (over 50 wards), no solution can be found in a reasonable time frame. This rules out the composite management of all resources by a single agent—the *CRA*. Therefore, a more sophisticated approach for allocating distributed resources has been tested. Responsibility for the global resources is split among a selected set of *SAs*, appoints each of them a *Resource Manager Agent* (*RMA*) of one shared resource. This approach transforms each non-binary resource constraint into a set of binary constraints which are managed by a selected *SA*.

The experimental evaluation of the three proposed protocols demonstrates a clear behavior. Adding concurrency to the protocol of the *CRA* improves its performance. When the *CRA* requests changes in the *SAs* schedules concurrently, the number of needed iterations is reduced by about 60%, see figure. The third proposed protocol is superior to the first two. Assigning *SAs* as Resources Managers, proves to be the best protocol. This way, 50% of the needed iterations for a good global solution are saved and about three times of the amount of the agents are kept busy computing alternative solutions, compared to the concurrent *CRA* protocol.

Experimental analysis of the implemented MAS system for the Soroka medical center provides the basis for the expected behavior of the proposed DisETP system. The results of our study indicate that the proposed framework has the potential to reduce the cost of transportation for the nurses scheduled by the hospital.

References

- Bellicha, A. (1993). Maintenance of solution in a dynamic constraint satisfaction problem. In *Proc. of applications of artificial intelligence in engineering VIII, Toulouse* (pp. 261–274).
- Bessière, C. (1991). Arc-consistency in dynamic constraint satisfaction problems. In *Proceedings of the ninth national conference on artificial intelligence, Anaheim, CA* (pp. 221–226).
- Bessière, C. (1992). Arc-consistency for non-binary dynamic csps. In *Proceedings of the 10th European conference on artificial intelligence, Vienna* (pp. 23–27).
- Bessière, C., Maestre, A., & Messeguer, P. (2001). Distributed dynamic backtracking. In *Workshop on distributed constraint of IJCAI01*.
- Burke, E., De Causmaecker, P., & Vanden Berghe, G. (1998). A hybrid tabu search algorithm for the nurse rostering problem. In *SEAL '98, Canberra, Australia* (pp. 187–194).
- Burke, E., Cowling, P., De Causmaecker, P., & Vanden Berghe, G. (2001). A memetic approach to the nurse rostering problem. *International Journal of Applied Intelligence*, 15(3), 199–214.
- Burke, E., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, 6, 1–14.
- Caseau, Y., Giulio, P., & Levenez, E. (1993). A deductive and object-oriented approach to a complex scheduling problem. In *Lecture notes in computer science: Vol. 760. Deductive and object oriented databases, Phoenix, AZ* (pp. 67–80).
- De Causmaecker, P., & Vanden Berghe, G. (2002). Relaxation of coverage constraints in hospital personnel rostering. In *Proceedings of the 4th international conference on the practice and theory of automated timetabling PATAT'02, Ghent, Belgium* (pp. 129–147).
- Dechter, R. (2003). *Constraints processing*. New York: Morgan Kaufman.
- Dechter, R., & Dechter, A. (1988). Belief maintenance in dynamic constraint networks. In *Proceedings of the seventh annual conference of the American association of artificial intelligence* (pp. 37–42).
- Faltings, B., & Macho-Gonzalez, S. (2002). Open constraint satisfaction. In P. V. Hentenryck (Ed.), *Lecture notes in computer science: Vol. 2470. Proceedings of principles and practice of constraint programming CP 2002, Ithaca, NY, USA* (pp. 356–370).
- Glover, F., & Laguna, M. (1997). *Tabu search*. Dordrecht: Kluwer Academic.
- Gomes, C., Selman, B., & Kautz, H. (July 1998). Boosting combinatorial search through randomization. *Proceedings of AAAI-98, Madison, WI, July 1998* (pp. 281–331).
- Grobner, M., & Wilke, P. (2001). Optimizing employee schedules by a hybrid genetic algorithm. In *Proceedings of the evo workshops* (pp. 463–472). Berlin: Springer.
- Hamadi, Y., & Bessière, C. (1998, August). Backtracking in distributed constraint networks. In *Proc. ECAI-98, Brighton* (pp. 219–223).
- Kragelund, L. (1997). Solving a timetabling problem using hybrid genetic algorithms. *Software: Practice and Experience*, 27, 1121–1134.
- Kragelund, L., & Kabel, T. (1998). *Employee timetabling*. M.Sc. Thesis, IR-129.
- Lamma, E., Mello, P., Milano, M., Cucchiara, R., Gavaneli, M., & Piccardi, M. (1999). Constraint propagation and value acquisition: why we should do it interactively. In T. Dean (Ed.), *Proc. of intern. joined conference of artificial intelligence JCAI 99* (pp. 468–477). Stockholm: Morgan Kaufmann.
- Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of third international joint conference on autonomous agents and multiagent systems (AAMAS 2004)* (pp. 438–445). IEEE Computer Society.
- Meisels, A., & Kaplansky, E. (2004). Iterative restart technique for solving timetabling problems. *European Journal of Operational* 153, 41–50.
- Meisels, A., & Kaplansky, E. (2002). Scheduling agents—distributed employee timetabling (detp). In *Proceedings of the 4th international conference on the practice and theory of automated timetabling PATAT'02, Ghent, Belgium, July 2002* (pp. 166–180).
- Meisels, A., & Lusternik, N. (1997, August). Experiments on networks of employee timetabling problems. In *Lecture notes in computer science: Vol. 1408. Pract. theo. autom. timetab. II, Toronto, Canada* (pp. 130–141).
- Meisels, A., & Schaerf, A. (2003). Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence*, 39, 41–59.
- Messeguer, P., & Jimenez, M. A. (2000). Distributed forward checking. In *Proc. CP-2000 workshop on distributed constraint satisfaction, Singapore, 22 September 2000*.
- Nguyen, T., & Deville, Y. (1998). A distributed arc-consistency algorithm. *Science of Computer Programming*, 30, 227–250.
- Petrovic, S., Beddoe, G., & Vanden Berghe, G. (2002). Storing and adapting repair experiences in employee rostering. In *Proceedings of the 4th international conference on the practice and theory of automated timetabling PATAT'02, Ghent, Belgium* (pp. 148–165).

- Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9, 268–299.
- Prosser, P. (1994). Binary constraint satisfaction problems: some are harder than others. In *Proceedings of the 11th European conference on artificial intelligence, Amsterdam* (pp. 95–99).
- Schaerf, A., & Meisels, A. (1999). Solving employee timetabling problems by generalized local search. In *Proc. Italian AI ass.* (pp. 493–502).
- Silaghi, M. C., Stefan, S., Sam-Haroud, D., & Faltings, B. (2001). Asynchronous search for numeric DiSC-SPS. In *CP2001, Paphos, Cyprus*.
- Solotorevsky, G., & Gudes, E. (1996). Solving a real-life nurses timetabling and transportation problem using distributed csp techniques. In *Proceedings of the second international conference on principles and practice of constraint programming (CP96), Cambridge, MA, USA*.
- Solotorevsky, G., Gudes, E., & Meisels, A. (1996). Modeling and solving distributed constraint satisfaction problems (dcsp). In *Constraint processing-96, New Hampshire*.
- Solotorevsky, G., Shimony, E., & Meisels, A. (1998). Csps with counters: a likelihood based heuristic. *Journal of Experimental & Theoretical Artificial Intelligence*, 10, 117–129.
- Tsang, E. (1993). *Foundations of constraint satisfaction*. New York: Academic.
- Verfaillie, G., & Schiex, T. (1994a). Dynamic backtracking for dynamic constraint satisfaction problems. In *Proceedings of the European conference on artificial intelligence workshop on “constraint satisfaction issues raised by practical applications”*.
- Verfaillie, G., & Schiex, T. (1994b). Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the twelfth conference of the American association of artificial intelligence* (pp. 307–312).
- Yokoo, M. (2001). *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Berlin: Springer.
- Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). Distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10, 673–685.