Distributed University Timetabling with Multiply Sectioned Constraint Networks

Yang Xiang
University of Guelph, Canada

Wanling Zhang
SAP Canada Labs, Canada

Abstract

Although timetabling has long been studied through constraint satisfaction based techniques, along with many alternatives, only recently work has been reported where distributed timetabling problems (DisTTPs) was studied as distributed constraint satisfaction problems (DisCSPs). We present an alternative method for solving DisTTPs based on multiply sectioned constraint networks (MSCNs). The proposed solution has several distinguishing features: Unlike the existing algorithms for DisCSPs whose worst-case time complexities are exponential, the algorithm suite based on MSCNs is efficient when the network topology is sparse. Unlike the existing DisTTP algorithm where a central agent is needed, there is no need for a central agent in the proposed solution. Unlike the existing DisTTP algorithm where partial timetables of other agents must be disclosed to the central agent, the proposed method keeps partial timetables of all agents private. We report our preliminary experimental result on distributed university timetabling problems (DisUTTPs).

Multiagent system, Multiply Sectioned Constraint Networks, Distributed Constraint Satisfaction, Distributed University Timetabling.

Introduction

The timetabling problem (Schaerf 1999; Rossi-Doria *et al.* 2003; McCollum 2007) deals with scheduling of a set of lectures within a prefixed time period (typically a week) subject to a set of constraints involving instructors, students and space. Often, lectures are managed by different departments within an institution. The problem can be solved by centralized timetabling, where constraints are collected centrally from departments and space is centrally managed. A timetable that satisfies all constraints will be produced centrally, imposed onto departments, and space necessary to execute the timetable is allocated. Collecting departmental constraints centrally is subject to the communication cost, the cost to translate local knowledge into a common format, and time delay. It is often undesirable to centralize local constraints for privacy reasons.

Distributed timetabling offers another alternative. In such a setting, space is managed by individual departments. Each department timetables its own lectures. However, due to interdependency of some lectures managed by different departments (e.g., a lecture to be attended by students from multiple departments), timetables of departments involved must be coordinated. Solved properly, distributed timetabling avoids communicating and translating local constraints centrally, shortens scheduling process, and protects departmental privacy.

Although timetabling has long been studied through constraint satisfaction based techniques, along with many alternatives (Werra 1985; Schaerf 1999; Rossi-Doria *et al.* 2003), only recently work has been reported where DisTTPs were studied as DisCSPs (Meisels & Kaplansky 2002). The solution is obtained by a multiagent system consisting of one *scheduling agent* (SA) per department and an additional *central agent* (CA). SAs are responsible to propose departmental timetables that satisfy their local constraints. CA collects proposed departmental timetables and checks against global constraints. If violated, CA instructs SAs to revise departmental timetables. Experimental study was conducted in nurse timetabling. No general completeness or complexity analysis of the algorithm is reported.

In the more general quest for solving DisCSPs, most existing algorithms are extensions of CSP algorithms based on backtracking or iterative improvement. These include ABT (Yokoo *et al.* 1992; Silaghi & Faltings 2005; Bessiere *et al.* 2005), AFC (Meisels & Zivan 2007), AWC (Yokoo *et al.* 1998), and DBA (Yokoo & Hirayama 1996; Hirayama & Yokoo 2005). Among them, ABT, AFC and AWC guarantee completeness. Although they perform well on average problem cases, their worst-case time complexities are exponential. This means that for any problem, before search starts, no time bound better than exponential can be pre-specified.

In this work, we present an alternative method for solving DisTTPs based on MSCNs (Xiang & Zhang 2007). The proposed solution has several distinguishing features: Unlike the existing algorithms for solving DisCSPs whose worst-case time complexities are exponential, the algorithm suite based on MSCNs is efficient when the network topology is sparse. Unlike the existing DisTTP algorithm (Meisels & Kaplansky 2002) where CA is needed, there is no need for a central agent in the proposed solution. Unlike the existing DisTTP algorithm where partial timetables of other agents must be disclosed to CA, the proposed method keeps partial timetables of all agents private. We report our preliminary experimental result on DisUTTPs.

Background on MSCNs

We present background on representation of a DisCSP as an MSCN and on how to solve the DisCSP using a compiled representation of the MSCN (Xiang & Zhang 2007). How to model a DisUTTP as an MSCN will be presented in a subsequent section. We refer to principals involved in a DisCSP as *departments* because the semantics is obvious in the con-

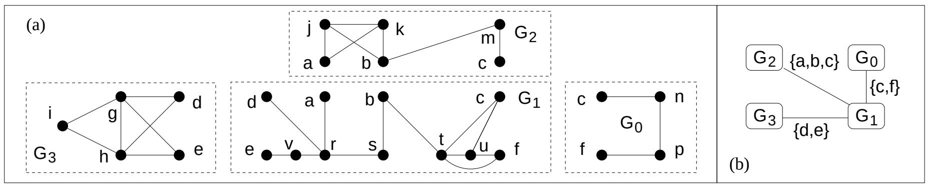

Figure 1: Primal graphs of subnets (a) and hypertree (b) of a trivial example MSCN.

text of DisTTPs and the application focus of this work is on DisUTTPs. We assume that each department is aided by an agent and the agents form a multiagent system.

An MSCN encodes a DisCSP in a problem *domain* consisting of a set V of variables and constraints specified over subsets of V. Each department/agent is concerned with a subset $V_i \subset V$ (called a *subdomain*) and a subset Λ_i of constraints specified over V_i. A subdomain and constraints over it are encoded as a *constraint subnet* $\mathcal{R}_i = (V_i, \Lambda_i)$. Each node in the subnet is labeled by a variable in V_i. For each constraint $R_X \in \Lambda_i$ over a set X of variables, elements of X are pairwise connected in the subnet. The subnet is embedded in the agent that represents the corresponding department. Note that $\mathcal{R}_i$ can be equivalently represented by a *primal graph* $G_i = (V_i, E_i)$, where E_i is the set of links determined by Λ_i. Without confusion, we will refer to both $\mathcal{R}_i$ and G_i by *subnet*.

Interdependencies between departments i and j are encoded through overlapping of their subdomains (i.e., $V_i \cap V_j \neq \emptyset$) and constraints over $V_i \cap V_j$. Agents solve the DisCSP by exchanging messages about partial solutions over these intersections. For such message passing to guarantee finding a solution if one exists and terminating if none exists, subdomains (and agents) are organized into a *hypertree*. Each hypernode is labeled by a subdomain and is associated with a corresponding agent. Each hyperlink is labeled by the intersection of two subdomains connected and is called an *agent interface*. The hypertree is so organized that the intersection of any two subdomains is contained in each hypernode on the hyperpath between them. Fig. 1 (a) shows subnets of an MSCN for a four-department DisCSP. The hypertree as well as agent interfaces are shown in (b).

To solve a DisCSP effectively by message passing among agents, each subnet is compiled into a cluster tree, called a *junction tree* (JT). For each link in each subnet, the corresponding constraint is assigned to a cluster in the JT. Based on assigned constraints, illegal configurations in the space of each cluster are eliminated locally. Each agent interface is also compiled into a cluster tree, called a *linkage tree* (LT), and JTs are linked through clusters of the LT (called *linkages*). The resultant runtime representation is a *linked junction forest* (LJF) (Xiang 2002), as illustrated in Fig. 2. The compilation of an MSCN to LJF is efficient.

Constraint propagation is performed through two rounds of message passing in the LJF. An arbitrary agent is selected as the root of the hypertree. In the first round, messages flow from leaf agents towards the root agent. At each agent, local constraint propagation is first performed so that its JT is arc-consistent, taking into account the constraints received from

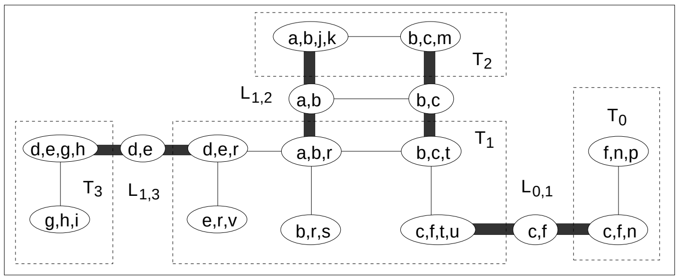

Figure 2: The LJF compiled from MSCN in Fig. 1.

child agents. It then passes the updated constraints on linkages to the parent agent. At the end of the first round, each parent agent is arc-consistent relative to each child agent. This round of message passing succeeds if there exists a solution for the DisCSP. Otherwise, the message passing terminates early and signifies that the DisCSP has no solution.

In the second round, messages flow from the root agent towards leaf agents. Each agent receives from the parent agent a configuration over each of their linkages. Taking into account of these messages, the agent performs local constraint propagation to determine the configuration for each cluster in its JT as well as that for each linkage with each child agent. It then sends the configurations over linkages to child agents. At the end of the second round, each agent will have determined the value for each variable in its subdomain. The collection of these value assignments constitutes a solution to the DisCSP. Hence, the algorithm suite is complete. Furthermore, the computational complexity is $O(n\ t\ k^{2q})$ (Xiang & Zhang 2007), where n is the number of agents, t the maximum number of clusters in a subnet JT, q the maximum size of clusters, and k binds cardinalities of spaces for variables in V. Hence, the computation is efficient when the size of the largest cluster is upper-bounded.

The DisUTTP Problem

The DisUTTP problem used in our experimental study is as follows: A university has a number of departments, each of which offers a number of courses to students in a given semester. The semester is divided into several weeks. Courses are scheduled for one week and the timetable is repeated for each week. Each week is divided into a set of prefixed time slots. Each course consists of two or more lectures per week and each lecture lasts for one time slot. Each department is allocated with a given number of lecture rooms. Scheduling is subject to several types of constraints.

1. No two lectures can be offered in the same room at the same time slot.
2. Lectures offered by the same instructor cannot be scheduled at the same time slot.
3. Lectures of the same course cannot be offered at the same time slot.
4. If two courses are to be taken by students in the same semester, then their lectures cannot be scheduled at the same time slots.

The last type of constraints can arise primarily due to two reasons: At a given semester, students in a given department may be required to take certain courses by the departmental curriculum. Such restriction is curriculum-imposed. The restriction is known well before timetabling is to be performed and must be respected by timetabling.

Alternatively, a given student or a group of students may choose to take a set of courses together in a given semester even though it is not required by the curriculum. Such preference is student-imposed. Most universities do not consider such preference in timetabling (McCollum 2007). To the contrary, students must pick among timetabled courses such that what they take are not in time-conflict. In light of this practice, we assume that the last type of constraints above is curriculum-imposed. If a student-imposed preference is general enough relative to a large group of students of the same year in their program, we assume that such preference has been communicated to the department and incorporated into the curriculum.

Courses in each department is scheduled independently into its available rooms over available time slots, unless there exist inter-departmental constraints. The type 4 constraints affect both intra and inter-departmental scheduling. For instance, students in department $Dept_1$ are required to take courses Crs_1 and Crs_2 in a semester, while students in $Dept_2$ are required to take Crs_2 and Crs_3 in the same semester. Hence, Crs_2 must be scheduled identically at both departments, its lectures cannot be scheduled into the same time slots with lectures of Crs_1 at $Dept_1$, and nor be scheduled into the same time slots with lectures of Crs_3 at $Dept_2$. That is, timetabling of Crs_1, Crs_2 and Crs_3 at both departments must be coordinated.

Given the above problem description, the objective is to timetable lectures of courses in each department into the given set of rooms and the given set of time slots such that all intra and inter-departmental constraints are satisfied.

Representing DisUTTP as MSCN

To solve the DisUTTP, we encode it into an MSCN. As the weekly timetable is repeated (this is the case in the institution of the first author), we focus on scheduling of lectures in a week over predetermined time slots.

We choose to represent each lecture during the week as one variable x. The collection of such variables forms the domain V of the MSCN. Each variable x is associated with a tuple $(C_x, I_x, t_x, r_x, CO_x)$. C_x is a course ID specifying the course that lecture x belongs to. Each course is offered by a unique department. I_x is an instructor ID specifying the instructor who has been assigned to teach the course. Element t_x is itself a variable, representing the time slot that lecture x will be scheduled into. Time slot t_x is associated with its space D_{t_x}. In the most general case, D_{t_x} would include all time slots of the week. Element r_x is also a variable, representing the room where lecture x will be offered. Room r_x is associated with the space D_{r_x}. In the most general case, D_{r_x} would include all lecture rooms allocated to the department that manages the course C_x. Finally, CO_x ($C_x \notin CO_x$) is a set of IDs for courses required to be taken with C_x.

Given the associated tuple $(C_x, I_x, t_x, r_x, CO_x)$, the space D_x of variable x is defined as the Cartesian product $D_x = D_{t_x} \times D_{r_x}$, representing all possible ways in which lecture x may be timetabled. Such a representation, although general, suffers from high computational cost. Since D_{t_x} includes all time slots of the week and D_{r_x} includes all lecture rooms allocated to the relevant department, the cardinality $|D_x| = |D_{t_x}| \times |D_{r_x}|$ can be large. Let the largest space cardinality for variables in V be denoted by α. Furthermore, any two variables x and y whose spaces D_x and D_y overlap are constrained due to type 1 constraints. That is, they should be directly connected in the MSCN. As the result, in each constraint subnet (corresponding to lectures managed by a department/agent), every pair of variables is directly connected. Let the largest number of lectures managed by a department be denoted by β. The computational complexity to solve the MSCN is proportional to α^β (see section on Background). Based on this representation, solving the resultant MSCN will be intractable.

To avoid the intractability, we explore preassignment of time slots and rooms to lectures. That is, we will heuristically reduce the spaces D_{t_x} and D_{r_x} for each variable x, and effectively reduce the cardinality of the space $D_x = D_{t_x} \times D_{r_x}$. As a result, α will be reduced. Furthermore, as the space of each variable x is reduced, less pairs of variables will have overlapping spaces. This results in a more sparse MSCN graphical structure and smaller clusters in the compiled LJF. As mentioned in the background section, if the size of the largest cluster in the LJF is upper-bounded, then solving the MSCN is efficient. Preassignment has been discussed in the literature (see, for example, (Werra 1985; Schaerf 1999)) as a practical need to be addressed in timetabling. In this work, we approach preassignment from a different perspective for efficiency improvement in solving DisUTTPs based on MSCNs.

Once domain variables are determined, constraints between them are specified in each constraint subnet. This amounts to connecting each pair of variables if there exists a constraint between them. Below, we elaborate on each type of constraints presented informally above.

1. Room-slot constraint: No two lectures can be offered in the same room at the same time slot. If lecture variables x and y satisfy $D_x \cap D_y \neq \emptyset$, then they are subject to a room-slot constraint. A legal configuration over $\{x, y\}$ must satisfy the condition: either $r_x \neq r_y$ or $t_x \neq t_y$.
2. Instructor constraint: Lectures offered by the same instructor cannot be scheduled at the same time slot. If

lecture variables x and y satisfy $I_x = I_y$, then they are subject to an instructor constraint. A legal configuration over $\{x, y\}$ must satisfy the condition: $t_x \neq t_y$.

3. Course constraint: Lectures of the same course cannot be offered at the same time slot. If lecture variables x and y satisfy $C_x = C_y$, then they are subject to a course constraint. A legal configuration over $\{x, y\}$ must satisfy the condition: $t_x \neq t_y$.

4. Course group constraint: If two courses are required to be taken by students in the same semester, then their lectures cannot be scheduled at the same time slots. If lecture variables x and y satisfy $C_x \in CO_y$, then they are subject to a course group constraint. Note that if $C_x \in CO_y$, then it must be the case $C_y \in CO_x$. A legal configuration over $\{x, y\}$ must satisfy the condition: $t_x \neq t_y$.

The following algorithm describes the construction of a constraint subnet associated with an agent.

Algorithm 1 (SetSubnet)
Input: The set of lectures V_i for all courses to be taken by students in i'th department.
Output: A constraint subnet $\mathcal{R}_i = (V_i, \Lambda_i)$.

```
begin
    $\Lambda_i = \emptyset$;
    for each pair of $x \in V_i$ and $y \in V_i$, do
        if $D_x \cap D_y \neq \emptyset$,
            add $\langle x, y\rangle$ to $\Lambda$;
            label $\langle x, y\rangle$ as a room-slot constraint;
        if $I_x = I_y$,
            if $\langle x, y\rangle \notin \Lambda$, add $\langle x, y\rangle$ to $\Lambda$;
            label $\langle x, y\rangle$ as an instructor constraint;
        if $C_x = C_y$,
            if $\langle x, y\rangle \notin \Lambda$, add $\langle x, y\rangle$ to $\Lambda$;
            label $\langle x, y\rangle$ as a course constraint;
        if $C_x \in CO_y$,
            if $\langle x, y\rangle \notin \Lambda$, add $\langle x, y\rangle$ to $\Lambda$;
            label $\langle x, y\rangle$ as a course group constraint;
    return $\mathcal{R}_i = (V_i, \Lambda_i)$;
end
```

Note that a given pair of lectures may be labeled with multiple constraints, e.g., an instructor constraint and a room-slot constraint. Next, we consider inter-department constraints and their representation. The following cases may arise.

1. An instructor may be jointly appointed by two or more departments and hence may teach courses offered by each department involved. We represent lectures for these courses in the subnets of all relevant departments so that instructor constraints over these lectures can be respected in timetabling.

2. A lecture room may be shared by two or more departments and hence courses from each department involved may be timetabled into the room. We represent those lectures, whose preassigned rooms include such rooms, in subnets of all relevant departments so that room-slot constraints over these lectures can be respected in timetabling.

3. Students in a given department may be required to take courses offered by a different department. For instance, Computer Science students are required to take Calculus offered by Mathematics department. Lectures of courses to be offered by a department but to be taken also by students in another department will be represented in subnets of both departments. Therefore, course group constraints affecting these lectures will be respected in timetabling.

Based on the above representational scheme, these lecture variables as well as the constraints among them will form the agent interfaces in the MSCN between relevant subnets.

Experimental Results

We report results from preliminary experiments. The experimental setup consists of five departments. Each week is divided into 15 time slots. Each department offers 20 lectures per week by 15 instructors. Each department manages four rooms exclusively plus one room shared by all departments. The experimental implementation is based on WEBWEAVR (Xiang), a Java-based toolkit for graphical models.

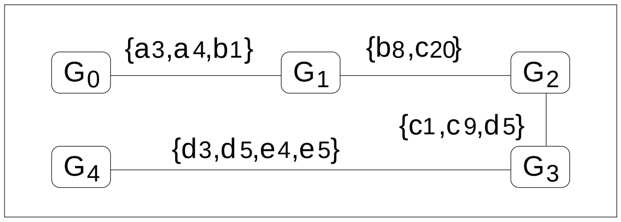

Figure 3: The hyperchain of MSCN for the first DisUTTP.

Two different DisUTTP problems were solved. For the first problem, the inter-departmental course dependency results in a hyperchain agent organization, shown in Fig. 3. The five agents are indexed as $A_0, \ldots, A_4$. Each hypernode in the hyperchain is associated with an agent A_i and is labeled in the figure by the primal graph G_i of its subnet. Agent interfaces are illustrated in terms of lecture variables contained in each interface.

Table 1 illustrates the subdomain of A_0. For each lecture in the subdomain, its course ID, instructor ID, time slot preassignment, and room pre-assignment are shown in the first four columns. If the lecture is in the agent interface, the agent interfaced with is shown in the fifth column. The type of interdepartmental dependency that causes the inclusion of the lecture in the interface is indicated in the last column.

In the second DisUTTP problem, the agent organization is a more general hypertree, as shown in Fig. 4.

Table 2 summarizes key parameters of the MSCN and the LJF compiled from it for the first DisUTTP problem. Each row characterizes the subnet and its JT of one agent. The number of variables in each subnet is shown in the second column. The total number of lectures timetabled is 100. The sparseness of each subnet is indicated by the number of links

Table 1: Subdomain of agent A_0 in the first DisUTTP.

Course ID	Instructor ID	Time Slot	Room	Agent Interfaced	Interface Constraint
a1	1	2,6,12	2,4		
a2	2	1,9,11	1,2		
a3	3	3,7,13	2,3,9	A_1	room
a4	4	5,8,15	3,4,9	A_1	room
a5	5	4,8,14	3,4		
a6	6	3,8,13	1,3		
a7	7	5,9,15	2,3		
a8	8	2,7,12	1,4		
a9	9	1,10,11	1,2		
a10	10	4,10,14	2,3		
a11	11	1,6,11	1,4		
a12	11	1,6,11	2,4		
a13	12	2,7,12	1,4		
a14	12	2,7,12	1,3		
a15	13	3,8,13	2,4		
a16	13	3,8,13	1,2		
a17	14	4,9,14	2,3		
a18	14	4,9,14	1,3		
a19	15	5,10,15	3,4		
a20	15	5,10,15	2,4		
b1	16	5,6,12	6,7,9	A_1	room

Table 2: Key parameters of MSCN and its LJF for the first DisUTTP.

Agent	#Variable	#Link	#Cluster	Max Cluster Size	Max Cluster Space
A_0	21	46	12	6	$6^5 \times 9$
A_1	23	54	15	6	$6^5 \times 9$
A_2	22	53	16	6	6^6
A_3	24	57	16	6	6^6
A_4	22	55	15	6	6^6

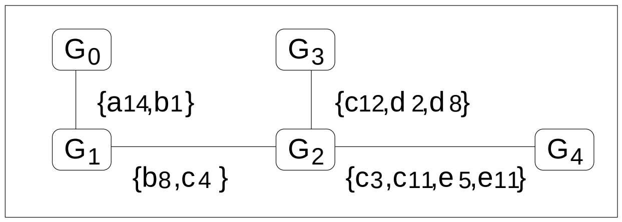

Figure 4: The hypertree of MSCN for the second DisUTTP.

in its primal graph shown in the third column. The remaining columns describe local JTs compiled from subnets. The number of clusters in each JT is indicated in the fourth column. The fifth column indicates the size of the largest cluster in each JT. The maximal number of configurations per cluster in each JT is indicated in the last column. As indicated in the previous section, computational complexity for solving a DisCSP is upper bounded by the maximum number of configurations per cluster in the LJF of its MSCN. Table 3 summarizes key parameters of the MSCN and its compiled LJF for the second DisUTTP problem.

As described in the background section, solving an MSCN consists of two rounds of constraint propagation along the hypertree of its LJF. It can be shown that after the first round of propagation, the JT in each agent is internally arc-consistent. Furthermore, each parent agent is arc-consistent relative to each child agent. This intra-agent arc-consistency and inter-agent arc-consistency guarantee backtracking-free solution of the MSCN through the second round of propagation. The dual arc-consistency holds even though JTs are not internally arc-consistent before the first round of propagation. Although the internal arc-consistency before the first round is unnecessary, it can reduce the number of legal configurations in each cluster in a JT. As seen from Tables 2 and 3, the maximum number of configurations per cluster for the two problems are sufficiently large. To reduce the space consumption during first round of propagation, we choose to run one round of internal constraint propagation in the JT of each agent, termed *preprocessing*.

Table 3: Key parameters of MSCN and its LJF for the second DisUTTP.

Agent	#Variable	#Link	#Cluster	Max Cluster Size	Max Cluster Space
A_0	21	49	11	6	6^6
A_1	22	54	14	6	$6^5 \times 9$
A_2	25	56	18	6	$6^5 \times 9$
A_3	21	53	14	6	6^6
A_4	22	58	14	6	6^6

Each DisUTTP problem is solved by a multiagent system, where five agents are distributed in four HP p920 Workstations (Intel Pentium-4 2.4GHz CPU and 512MB memory). Table 4 shows the runtime for solving each problem. The runtime for preprocessing and the two rounds of propagation are shown in the second and third columns, respectively.

Table 4: Runtime for solving the two DisUTTP problems.

Agent Organization	Preprocessing	Propagation
hyperchain	5min 45sec	8min 23sec
hypertree	5min 23sec	10min 46sec

Remarks

We presented an alternative method for solving DisUTTPs based on MSCNs. Existing DisCSP algorithms based on ABT, AFC, AWC and DBA have exponential worst-case complexity. No time bound better than exponential can be given before problem solving starts. The MSCN based algorithm is efficient when the largest cluster in its LJF is upper-bounded in size. As long as sparseness of the MSCN remains the same, complexity of the algorithm grows linearly with the number of agents and sizes of subnets. The size of the largest cluster is known after LJF compilation. Hence, for sparse MSCNs, a linear time bound is available before problem solving starts. When clusters in LJF are too large, the MSCN can be be made sparser by relaxing some soft constraints before recompilation and problem solving.

The existing DisTTP algorithm based on SA/CA needs a central agent CA to whom partial timetables of other agents must be disclosed. The MSCN based algorithm selects an arbitrary agent as root who is identical function-wise to any other agent. The only timetables exchanged between agents are those over their interfaces - lectures shared by multiple departments, and hence better agent privacy is achieved.

Investigation of alternative DisUTTP problems for further improvement of efficiency and more extensive experimental study are underway.

Acknowledgement

Financial support to the first author through Discovery Grant from NSERC, Canada is acknowledged.

References

Bessiere, C.; Maestre, A.; Brito, I.; and Meseguer, P. 2005. Asynchronous backtracking without adding links: a new member in the ABT family. *Artificial Intelligence* 161(1-2):7–24.

Hirayama, K., and Yokoo, M. 2005. The distributed breakout algorithms. *Artificial Intelligence* 161(1-2):89–116.

McCollum, B. 2007. A persepctive on bridging the gap between theory and practice in university timetabling. In *Practice and Theory of Automated Timetabling VI, Springer Lecture Notes in Computer Science Vol 3867*. 3–23.

Meisels, A., and Kaplansky, E. 2002. Scheduling agents - distributed timetabling problems (DisTTP). In *Proc. 4th Conf. on Autom. Timetabling (PATAT-2002), LNCS 2740*, 166–180.

Meisels, A., and Zivan, R. 2007. Asynchronous forward-checking for DisCSPs. *Constraints* 12(1):131–150.

Rossi-Doria, O.; Sampels, M.; Birattari, M.; Chiarandini, M.; Dorigo, M.; Gambardella, L. M.; Knowles, J.; Manfrin, M.; Mastrolilli, M.; Paechter, B.; Paquete, L.; and Stutzle, T. 2003. A comparison of the performance of different metaheuristics on the timetabling problem. In Burke, E., and Causmaecker, P., eds., *Practice and Theory of Automated Timetabling IV, LNCS 2740*. Springer.

Schaerf, A. 1999. A survey of automated timetabling. *Artificial Intelligence Review* 13:87–127.

Silaghi, M., and Faltings, B. 2005. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence* 161(1-2):25–54.

Werra, D. 1985. An introduction to timetabling. *European J. Operational Research* 19:151–162.

Xiang, Y., and Zhang, W. 2007. Multiagent constraint satisfaction with multiply sectioned constraint networks. In Kobti, Z., and Wu, D., eds., *Advances in Artificial Intelligence, LNAI 4509*. Springer-Verlag. 228–240.

Xiang, Y. WebWeavr-IV Research Toolkit. www.cis.uoguelph.ca/˜yxiang/.

Xiang, Y. 2002. *Probabilistic Reasoning in Multi-Agent Systems: A Graphical Models Approach*. Cambridge University Press, Cambridge, UK.

Yokoo, M., and Hirayama, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proc. 2nd Inter. Conf. on Multi-Agent Systems*, 401–408.

Yokoo, M.; Durfee, E.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proc. 12th IEEE Inter. Conf. on Distributed Computing Systems*, 614–621.

Yokoo, M.; Durfee, E.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowledge and Data Engineering* 10(5):673–685.