Optimal Multi-Agent Scheduling with Constraint Programming*

Willem-Jan van Hoeve[1], **Carla P. Gomes**[1], **Michele Lombardi**[2], and **Bart Selman**[1]
[1] Dept. of Computer Science, Cornell University, 4130 Upson Hall, Ithaca, NY 14853
[2] DEIS, University of Bologna, Viale Risorgimento 2, 40136, Bologna, Italy
{gomes,vanhoeve,selman}@cs.cornell.edu, mlombardi@lompa.it

Abstract

We consider the problem of computing optimal schedules in multi-agent systems. In these problems, actions of one agent can influence the actions of other agents, while the objective is to maximize the total 'quality' of the schedule. More specifically, we focus on multi-agent scheduling problems with time windows, hard and soft precedence relations, and a nonlinear objective function. We show how we can model and efficiently solve these problems with constraint programming technology. Elements of our proposed method include constraint-based reasoning, search strategies, problem decomposition, scheduling algorithms, and a linear programming relaxation. We present experimental results on realistic problem instances to display the different elements of the solution process.

Introduction

Multi-agent planning and scheduling problems arise in many contexts such as supply chain management, coordinating space missions, or configuring and executing military scenarios. In these situations, the agents usually need to perform certain tasks in order to achieve a common goal. Often the agents need to respect various restrictions such as temporal constraints and interdependency relations. Furthermore, depending on the application at hand, these problems may be subject to several uncertainties, for example the actual outcome and duration of executing a task, and changing environmental conditions. Multi-agent planning and scheduling problems are among the most difficult problems in Artifical Intelligence. While the centralized deterministic version is already NP-hard, the non-deterministic distributed version is even NEXP-complete (Bernstein *et al.* 2002).

In this work we present an efficient method to compute provably optimal solutions for deterministic multi-agent scheduling problems. Our method can be applied either in a centralized or in a distributed multi-agent setting. For example, one can use our method to compute an initial optimal centralized schedule for all agents together. In a distributed setting, it can be applied as a local scheduler to compute alternative schedules for individual agents, in which case 'optimality' is restricted to the local view of each agent. Finally, our solver may be applied in an experimental setting to evaluate distributed approaches with respect to an optimal solution.

Our approach is based on constraint programming technology. This has several advantages. First, it allows us to specify the problem in a rich modeling language, and to apply the corresponding default constraint-based reasoning. As we will see below, our model is very close to the original representation of the problem. Second, in the constraint programming framework we can specify detailed search heuristics, tailored to the specific needs of the problem. In addition, we have implemented a problem decomposition scheme to further improve our search process. Third, we have implemented an "optimization constraint", based on a linear programming relaxation of the problem, to strengthen the optimization reasoning. Fourth, we optionally apply advanced scheduling algorithms, such as the edge-finding algorithm. The constraint-based structure of constraint programming allows us to implement all these technologies efficiently in one system.

In the following section we provide a detailed description of the problem class that is the subject of this paper. Thereafter, we present our constraint programming model. This is followed by a description of the solution process. Finally, we present extensive computational results.

Problem Description

The problems that we consider in this work consist of a set of *agents* that may execute certain *methods*. Each executed method contributes an amount of *quality* to a hierarchical objective function. Furthermore, the problems contain temporal constraints and interdependence relations that need to be respected. The goal is to find for each agent a schedule of methods to execute at a certain time, such that the total quality is maximized. To represent these problems, we make use of the modeling language TAEMS: a framework for Task Analysis, Environment Modeling, and Simulation (Horling *et al.* 1999). In fact, we consider a subset of this framework, called cTAEMS, which is particularly suitable to represent coordination problems (Boddy *et al.* 2007).

In cTAEMS, a problem is represented by *tasks* and *methods*, which are linked to each other in a hierarchical way. An example is depicted in Figure 1, consisting of 9 tasks and 13 methods. A *method* i is owned by a single agent $A[i]$.

*This version (January 24, 2007) has been submitted and is under review.

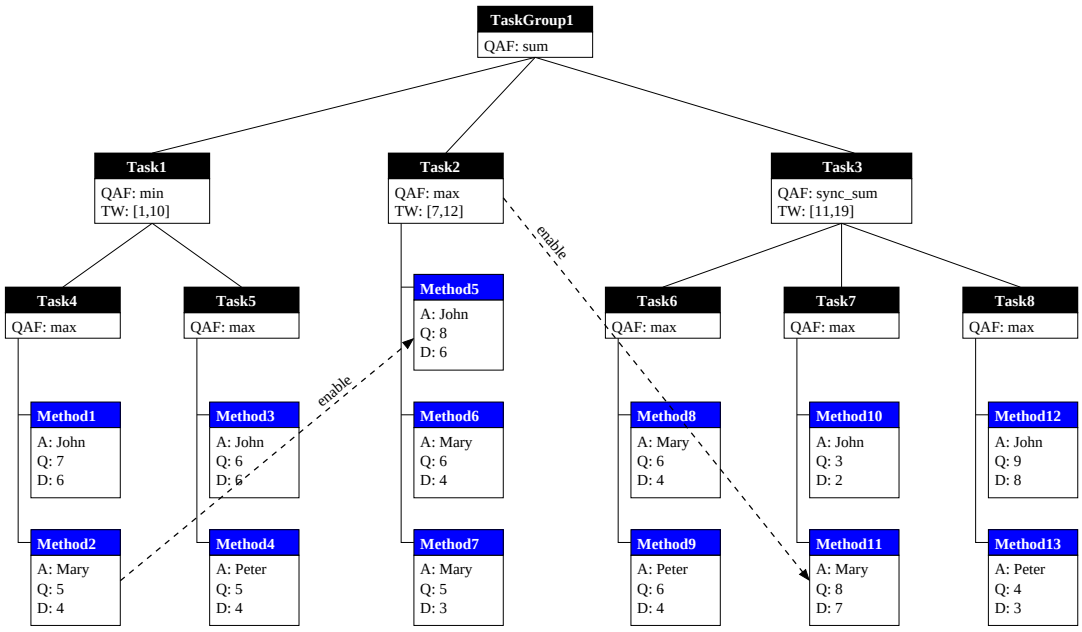

Figure 1: *Example of a cTAEMS task structure. QAF stands for quality accumulation function, TW for time window, A for agent, Q for quality, and D for duration.*

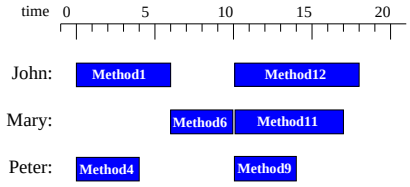

Figure 2: *The optimal schedule for the agents corresponding to the problem of Figure 1, with total quality 34.*

Each agent is restricted to execute at most one method at a time. If a method i is executed, it generates a certain quality $Q[i]$, while its execution takes a duration $D[i]$. A *task* is not owned by an agent, but serves to accumulate quality via its subtasks (or submethods). This is done via a *quality accumulation function*, or QAF. The possible QAFs are: *min*, *max*, *sum*, *sync-sum*, *exactly-one*, and *sum-and*. Here *min*, *max*, and *sum* represent the minimum, maximum, or sum, respectively. The *sync-sum* represents the sum of all subtasks (or submethods) that are synchronized, i.e., starting at the same time. The *exactly-one* restricts at most one subtask (or submethod) to have positive quality. The *sum-and* requires all subtasks (or submethods) to have positive quality, or none. The accumulation of quality only takes place after a method has been completed. The total quality of the problem is represented by the root task (called TaskGroup1 in Figure 1). The goal is to execute certain methods such that the quality of the root task is maximized. For example, Figure 2 presents an optimal solution for the problem in Figure 1.

The execution of a method takes place from its *start time* until its *end time*. The integer time representation is such that the duration includes the start and end time. For example, in Figure 2, Method6 starts at time 7, ends at time 10, and has a duration of 4. The start and end time of a task are inherited recursively by the start and end time of its children. Both methods and tasks may be subject to a *time window*, representing the earliest start time and latest end time (denoted by TW in Figure 1). Time windows also apply to the tasks and methods underneath a task. Hence, the time window of a method is defined by the intersection of the time windows of all tasks on the path from the method to the root task of the hierarchy.

Finally, there may exist *precedence relations* between tasks and/or methods. The possible precedence relations are: *enable*, *disable*, *facilitate*, and *hinder*. A precedence relation influences the execution of the target, proportional to the quality of the source at the time of execution of the target. An *enable* relation states that we may only execute the target from the moment that the source has positive quality. For example, in Figure 1 we may execute Method11 after Task2 has accumulated (some) positive quality. A *disable* relation forbids the execution of the target from the moment that the source has positive quality. The *facilitate* and *hinder* relation are 'soft' enabling or disabling relations, and have a coefficient $0 \leq c \leq 1$. Suppose the source can maximally accumulate maxQ quality. If the source has accumulated quality q at the time of execution of the target, the *facilitate* relations decreases the duration, and increases the quality, of the target with a factor $c \cdot q/\mathrm{maxQ}$. For a *hinder* relation, this factor is used to increase the duration and to decrease the quality of the target. By definition, *facilitate* and *hinder* relations only affect the duration of methods, while the

'duration' of a task is undefined.

As a final remark, cTAEMS allows the data to be specified by means of probability distributions rather than fixed numbers. In this work we restrict ourselves to deterministic data, however. If necessary, we replace the distributions by their minimum, maximum, or expected value.

Related Work

In the last decade there has been an increasing interest in centralized and distributed approaches to solve multi-agent planning and scheduling problems. In the context of distributed multi-agent systems representable with cTAEMS, several approaches have been developed. In those approaches, the main target is the coordination problem under changing environmental conditions. Naturally, each of the approaches also includes a 'scheduler' to compute and evaluate alternative solutions.

One approach, introduced by (Musliner *et al.* 2006), represents the non-deterministic cTAEMS problem as a Markov decision process (MDP). When computing a schedule (in fact a policy), the MDP is only partially 'unrolled' in order to keep the computational complexity under control. Another approach, proposed by (Szekely *et al.* 2006), applies a selective combination of different heuristic solution methods, including a partially-centralized solution repair, and locally optimized resource allocation. Finally, (Smith *et al.* 2006) represent cTAEMS problems as Simple Temporal Networks, and apply constraint-based reasoning to compute a solution to the deterministic version of the problem. However, also the latter method does not compute provably optimal solutions.

Although the non-deterministic cTAEMS problem can in theory be solved to optimality, for example by using Markov decision processes, in practice there exists no scalable such solver (to the best of our knowledge). For the non-deterministic cTAEMS problem, several methods exists, none of which is able to compute optimal solutions efficiently. A main contribution of this work is to present the first scalable solver that efficiently computes optimal solutions for the deterministic cTAEMS problem.

Constraint Programming Model

Variables

For each method i, we introduce the following *decision variables*: a binary variable x_i representing whether or not i is executed, and an integer variable start_i representing the start time of i. Together, they determine any potential schedule.

Furthermore, we make use of the following *auxiliary variables*. For each task i we introduce an integer variable start_i, representing its starting time. For each task or method i we introduce an integer variable end_i representing the end time of i, and a floating-point variable qual_i representing the quality of i. For each method i we further introduce a floating point variable dur_i representing its duration. Finally, for each precedence relation r we introduce a floating point variable factor_r representing the factor of r.

Temporal Constraints

The temporal constraint are expressed as follows. For each method i with with duration $D[i]$ and time window $[L, U]$:

$$\mathrm{dur}_i = D[i] \cdot x_i, \tag{1}$$

$$\mathrm{start}_i + \mathrm{dur}_i - 1 = \mathrm{end}_i, \tag{2}$$

$$\mathrm{start}_i \geq L, \tag{3}$$

$$\mathrm{end}_i \leq U. \tag{4}$$

In case method i is the target of precedence relations, we need to augment equation (1), which is described below.

Resource Constraints

The resource constraints ensure that the methods of an agent do not overlap (the agents correspond to a unary resource). For each agent a and each two different methods i and j with $A[i] = A[j] = a$, we state:

$$(\mathrm{start}_i > \mathrm{end}_j) \vee (\mathrm{start}_j > \mathrm{end}_i).$$

Alternatively, we can group together all non-overlapping constraints for each agent in one *UnaryResource* constraint. This allows to reason over all disjunctions together, for example using the edge-finding algorithm (Carlier & Pinson 1994; Vilim 2004). For each agent a, we then state

$$\textit{UnaryResource}(S_a, E_a, R_a),$$

where $S_a = \{\mathrm{start}_i \mid A[i] = a\}$ represents the start variables, $E_a = \{\mathrm{end}_i \mid A[i] = a\}$ the end variables, and $R_a = \{x_i \mid A[i] = a\}$ the "requirement" variables of methods i with $A[i] = a$.

Quality Accumulation

Next we consider the constraints to link together the quality of the tasks and the methods. For each method i with quality $Q[i]$ we state:

$$\mathrm{qual}_i = Q[i] \cdot x_i. \tag{5}$$

For each task t with subtasks $s_1, \ldots, s_k$ and quality accumulation function $f \in \{\textit{min}, \textit{max}, \textit{sum}\}$ we state:

$$\mathrm{qual}_t = f_{i=1,\ldots,k}\ \mathrm{qual}_{s_i}. \tag{6}$$

If the quality accumulation function is *sync-sum* we state:

$$\mathrm{qual}_t = \textstyle\sum_{i=1,\ldots,k} \mathrm{qual}_{s_i}, \tag{7}$$

$$(x_{s_i} = 1) \Rightarrow (\mathrm{start}_{s_i} = \mathrm{start}_t). \tag{8}$$

If the quality accumulation function is *exactly-one* we state:

$$\mathrm{qual}_t = \max_{i=1,\ldots,k} \mathrm{qual}_{s_i}, \tag{9}$$

$$(x_{s_1} = 1) + \ldots + (x_{s_m} = 1) \leq 1. \tag{10}$$

Finally, if the quality accumulation function is *sum-and* we state:

$$\mathrm{qual}_t = \textstyle\sum_{i=1,\ldots,k} \mathrm{qual}_{s_i}, \tag{11}$$

$$((x_{s_1} = 1) \wedge \ldots \wedge (x_{s_m} = 1)) \vee (\mathrm{qual}_t = 0). \tag{12}$$

In case the method or the task is the target of precedence relations, we need to augment the corresponding quality constraints. This is described below.

The objective function value is represented by the quality of the root of the search tree, $\mathrm{qual}_{\mathrm{root}}$. Hence, to maximize its quality, we add the 'constraint':

$$\textit{maximize}\ \mathrm{qual}_{\mathrm{root}}.$$

Precedence Relations

First we model the effect of precedence relations to the quality variables. If method i is the target of precedence relations $r_1, \ldots, r_m$, we replace equation (5) by:

$$\mathrm{qual}_i = \mathrm{factor}_{r_1} \cdot \ldots \cdot \mathrm{factor}_{r_m} \cdot Q[i] \cdot x_i.$$

If task t is the target of precedence relations $r_1, \ldots, r_m$, and has a quality accumulation function f, we replace the corresponding quality constraint (6), (7), (9), or (11) by:

$$\mathrm{qual}_t = \mathrm{factor}_{r_1} \cdot \ldots \cdot \mathrm{factor}_{r_m} \cdot f_{i=1,\ldots,k}\mathrm{qual}_{s_i}.$$

The duration variables are similarly updated. If method i is the target of *facilitate* and/or *hinder* relations $r_1, \ldots, r_m$, we replace equation (1) by:

$$\mathrm{dur}_i = \mathrm{factor}_{r_1} \cdot \ldots \cdot \mathrm{factor}_{r_m} \cdot D[i] \cdot x_i.$$

Next we describe how we model the factor variables. Recall that the precedence relations depend on the quality of the source at the start time of the target. For a precedence relation r from source i to target j (if applicable with coefficient c_r), we state:

$$\mathrm{factor}_r = (\mathrm{QExpr}(i, \mathrm{start}_j) > 0) \quad (\textit{enable}),$$
$$\mathrm{factor}_r = 1 + (c_r \cdot \mathrm{QExpr}(i, \mathrm{start}_j)/\mathrm{maxQ}_i) \quad (\textit{facilitate}),$$
$$\mathrm{factor}_r = 1 - (\mathrm{QExpr}(i, \mathrm{start}_j) > 0) \quad (\textit{disable}),$$
$$\mathrm{factor}_r = 1 - (c_r \cdot \mathrm{QExpr}(i, \mathrm{start}_j)/\mathrm{maxQ}_i) \quad (\textit{hinder}),$$

where $\mathrm{QExpr}(i, \mathrm{start}_j)$ is a recursive expression representing the quality of i at the start time of j, and maxQ_i is the maximum possible quality of i. The expression $\mathrm{QExpr}(i, \mathrm{start}_j)$ contains both temporal conditions and quality accumulation functions following from the subtree rooted at the source of the relation. For example, if r is an *enable* relation from method i to method j, we have

$$\mathrm{factor}_r = ((\mathrm{end}_i \geq \mathrm{start}_j) > 0).$$

Linear Programming Constraint

The objective function is composed of the functions *min*, *max*, *sum*, and complex nonlinear expressions following from the precedence relations. In order to potentially improve the optimization reasoning of the constraint programming solver, we have additionally implemented a redundant *optimization constraint*, based on a linear programming relaxation of the problem. We state the constraint as:

$$\textit{LP-constraint}(x, \mathrm{start}, \mathrm{end}, \mathrm{qual}_{\mathrm{root}}),$$

where x, start and end are shorthands for the arrays consisting of the variables x_i, start_i and end_i for all methods i, and $\mathrm{qual}_{\mathrm{root}}$ represents the quality variable of the root task. Each time the *LP-constraint* is invoked, it builds an internal linear programming model, taking into account the whole cTAEMS problem structure. Based on the continuous solution of this model, the upper bound of $\mathrm{qual}_{\mathrm{root}}$ is potentially improved. Furthermore, we apply *reduced-cost based filtering* to remove inconsistent values from the domains of the variables x_i (Focacci, Lodi, & Milano 1999).

Solution Techniques

Search Strategy

In constraint programming, the variable and value selection heuristics determine the shape of the search tree, which is usually traversed in a depth-first order. We have experimented with several different heuristics, and report here the most effective strategy, following from our experiments.

Our model consist of two sets of decision variables; the assignment variables x_i and the start variables start_i for each method i. We apply a two-phase depth-first search, consisting of a *selection phase* and a *scheduling phase*. In the selection phase, we assign all assignment variables. For this we use a greedy variable selection heuristic, i.e. choose first the variable x_i (for method i) for which the quality $Q[i]$ is highest (ties are broken lexicographically). As a value selection heuristic, we first choose value 0, and then value 1. Using this strategy, we start with an empty schedule that is gradually augmented with methods of high quality. If instead we would have chosen value 1 first, we often cannot schedule all selected methods.

In the scheduling phase we assign the start variables. As variable selection heuristic we choose first the variable with the smallest domain size (ties are again broken lexicographically). As value selection heuristic we choose first the minimum value in the domain.

Problem Decomposition

When certain parts of a problem are independent, one can decompose the problem and solve the parts independently. In constraint programming, independent subproblems are usually detected by means of the *constraint (hyper-)graph*. In the constraint graph of a model, the nodes represent the variables, while relations between variables (the constraints) are represented by (hyper-)edges. Independent subproblems are equivalent to connected components in the constraint graph, which thus represent distinct subsets of variables and their corresponding constraints. As the connected components can be found in linear time (in the size of the graph), problem decomposition can be very effective.

In our case, it suffices to build the constraint graph on the decision variables x_i and start_i for all methods i. In fact, we can simply group them together and create one node for each method i. We add an edge between two nodes i and j if there is a constraint involving method i and j. For example, if methods i and j belong to the same agent, and their time windows overlap, the non-overlapping constraint will place an edge between the nodes representing i and j. Naturally, at most one edge needs to be maintained for each pair of nodes.

Unfortunately, all decision variables are linked together via the objective function and the quality constraints. Hence, the constraint graph consists of one connected component, which prevents the application of problem decomposition. We have circumvented this restriction by decomposing the objective function more carefully. Namely, as we are maximizing, the arguments of the functions *sum* and *max* may be evaluated (and maximized) independently, while preserving optimality. For the *min* function this is not the case, because

	base	no decomposition	LP constraint	disjunctions	best known
optimal solutions	100%	98.8%	100%	100%	74.7% *
average time (s)	0.028	0.636	1.275	0.030	
median time (s)	0.02	0.02	0.43	0.02	
average backtracks	83.4	7114.1	77.0	82.4	
median backtracks	59	103	59	59	

* no proof of optimality

Table 1: *Computational results on problem set I. The base settings of our solver are: apply problem decomposition, omit the LP-constraint, and apply the UnaryResource constraint. Time limit is set to 300 seconds. The 'best known' column refers to the previously best known solutions.*

its arguments are dependent in case of maximization. Consequently, while building the constraint graph, we consider the quality accumulation functions of the objective function individually. When this function is a *min*, *sync-sum*, *exactly-one*, or *sum-and*, we add an edge between all methods underneath this function. We don't add any edges when the function is a *sum* or a *max*. Doing so, we are able to effectively decompose the problem in many cases.

Experimental Results

Our model is implemented in ILOG CP Solver 6.3, and uses the default constraints and corresponding domain filtering algorithms, where applicable. We have implemented our two-phase search strategy, the problem decomposition, and the *LP-constraint* within ILOG CP Solver 6.3. For the *LP-constraint* we use ILOG CPLEX 10.1 to solve the linear programming relaxation. The *UnaryResource* constraint applies the edge-finding algorithm of ILOG Scheduler 6.3. In all experiments, we apply a time limit of 300 seconds per instance.

We have performed experiments on problem instances originating from the DARPA program COORDINATORs. They represent realistic problem scenarios that are designed to evaluate all the different components of the problem. Problem set I consists of 2550 small to medium-sized instances, containing 8 to 64 methods (up to 128 decision variables), and 2 to 9 agents. We have used this set to evaluate the performance of our different solution strategies. Table 1 presents the computational results for this problem set, aggregating the results over all 2550 instances. We report the median and average time and number of backtracks, and the percentage of problems that could be optimally solved. We compare our results with the previously best known solutions, computed by a heuristic solver developed by Global InfoTek, Inc.[1] (unfortunately we were not able to determine the corresponding running times).

The column 'base' represents the results for our base settings: apply problem decomposition, omit the *LP-constraint*, and apply the *UnaryResource* constraint. With these settings we obtain the best results: all problems are solved to optimality, in the fastest time. The column 'no decomposition' shows the results if we omit the problem decomposition. In that case, only 98.8% of the instances are solved optimally (within the time limit of 300 seconds), while the time and number of backtracks increase drastically. The next column, 'LP constraint' shows the results when we activate the *LP-constraint*. Although we can solve all problems within the time limit, and the number of backtracks slightly decreases, the application of this constraint is too costly in terms of running time, for these instances. Finally, column 'disjunctions' shows the results when we replace the *UnaryResource* constraint with the disjunctive representation. In other words, the edge-finding algorithm is replaced with individual non-overlapping constraint. The results indicate that the two approaches are comparable for these instances.

The results in Table 1 indicate that the *LP-constraint* is not effective when applied to problem set I. We suspected that this is due to the relatively small number of *sum* functions in these problem instances. Hence, to investigate this further, we adapted problem set I by uniform-randomly replacing *min* and *max* quality accumulation functions by a *sum*. To avoid excessive problem decomposition we have also slightly increased the time windows (with a factor 0.15). The resulting problem set II has been solved with and without the *LP-constraint*. The computational results are presented in Table 2, indictating the benefit of the *LP-constraint* under the new circumstances. With the additional *LP-constraint* we can solve more problems optimally, provide better upper bounds, while the running time decreases.

	with LP	without LP
optimal solutions	84.8%	81.2%
best lowerbound	99.8%	92.1%
all instances:		
average time (s)	52.14	60.84
median time (s)	0.98	0.04
average backtracks	346,934	498,937
median backtracks	184	220
instances optimally solved with LP only:		
average time (s)	23.01	300.00
median time (s)	3.12	300.00
average backtracks	55,496	3,137,852
median backtracks	875	3,259,144

Table 2: *Computational results on problem set II. 'Without LP' corresponds to our base settings, while 'with LP' adds to our base settings the LP-constraint. Time limit is 300 seconds.*

[1] http://www.globalinfotek.com/

	Cornell	best known
optimal solutions	59%	26% *
best lowerbound	76%	50%

			Cornell			best known
instance	**#methods**	**#agents**	**obj. value**		**time (s)**	**obj. value**
big1001s	2250	100	2050.25	optimal	87.3	2050.25 *
big100EVA1	2250	100	2046.5	optimal	88.48	2046.50 *
big100EVA2	2250	100	2059.25	optimal	86.33	2059.25 *
big100EVA3	2250	100	2110	optimal	94.15	2110.00 *
big100EVA4	2250	99	2040.25	optimal	92.6	2040.25 *
big100EVA5	2250	99	2041.75	optimal	92.21	2041.75 *
big701s	1350	70	1314.12	lowerbound	300	1303.83
big70EVA1	1350	70	1312	optimal	83.51	1298.04
big70EVA2	1350	70	1310.81	lowerbound	300	1293.34
big70EVA3	1350	69	1321.87	lowerbound	300	1307.34
big70EVA4	1350	70	1309	lowerbound	300	1293.75
big70EVA5	1350	70	1298.38	optimal	259.05	1278.19

* no proof of optimality

Table 3: *Computational results for problem set III. Columns '#methods' and '#agents' denote the number of columns and agents, respectively. The three columns under 'Cornell' represent the objective function, optimality, and running time of our method, respectively. The 'best known' column refers to the previously best known solutions. Time limit is set to 300 seconds.*

Finally, we have tested our solver on large problem instances to test its robustness and scalability. For this we used problem set III, that consists of 46 medium-sized to large instances, containing 135 to 2250 methods (up to 4500 decision variables), and 8 to 100 agents. We have solved these problems with our base setting, and present the experimental results in Table 3. The upper part of Table 3 shows aggregated results over all instances, while the lower part presents detailed results on the largest instances. We compare our method with the previously best known solutions, which were computed using the method proposed by (Smith *et al.* 2006) (unfortunately we were not able to determine the corresponding running times). In many cases our solver is able to compute an optimal solution, and to improve or meet the current best solution. Moreover, even for the largest instances, our running times are often very fast. These results indicate that are method is both robust, efficient and scalable.

Conclusion

We have presented an efficient and scalable method to compute optimal solutions to multi-agent scheduling problems, based on constraint programming. We have focussed in particular on problems that are representable by the cTAEMS language. Our method can be applied to compute deterministic centralized optimal schedules to such problems, or it can be used as a local scheduler to compute alternative schedules for individual agents in a distributed setting. At the moment of writing, our solver is being used successfully for both purposes by researchers at Harvard University, SRI International, and Bar-Ilan University.

References

Bernstein, D.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27(4):819–840.

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R.; Vincent, R.; Long, A.; Kohout, B.; and Maheswaran, R. 2007. C_TAEMS Language Specification — Version 2.03.

Carlier, J., and Pinson, E. 1994. Adjustment of Heads and Tails for the Job-shop Problem. *European Journal of Operational Research* 78:146–161.

Focacci, F.; Lodi, A.; and Milano, M. 1999. Cost-Based Domain Filtering. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP '99)*, volume 1713 of *LNCS*, 189–203. Springer.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The Taems White Paper.

Musliner, D.; Durfee, E.; Wu, J.; Dolgov, D.; Goldman, R.; and Boddy, M. 2006. Coordinated Plan Management Using Multiagent MDPs. In *AAAI Spring Symposium on Distributed Plan and Schedule Management*, 73–80. AAAI Press.

Smith, S.; Gallagher, A.; Zimmerman, T.; Barbulescu, L.; and Rubinstein, Z. 2006. Multi-Agent Management of Joint Schedules. In *AAAI Spring Symposium on Distributed Plan and Schedule Management*, 128–135. AAAI Press.

Szekely, P.; Maheswaran, R.; Neches, R.; Rogers, C.; Sanchez, R.; Becker, M.; Fitzpatrick, S.; Gati, G.; Hanak, D.; Karsai, G.; and van Buskirk, C. 2006. An Examination of Criticality-Sensitive Approaches to Coordination. In *AAAI Spring Symposium on Distributed Plan and Schedule Management*, 136–142. AAAI Press.

Vilim, P. 2004. $O(n \log n)$ Filtering Algorithms for Unary Resource Constraint. In *Proceedings of CPAIOR 2004*, volume 3011 of *LNCS*, 319–334. Springer.