Constraint Programming for Distributed Planning and Scheduling

Carla P. Gomes and **Willem-Jan van Hoeve** and **Bart Selman**
Cornell University, Department of Computer Science
4130 Upson Hall
Ithaca, New York 14853–7501
{gomes,vanhoeve,selman}@cs.cornell.edu

Abstract

We present a constraint programming-based solver for distributed planning and scheduling problems with a hierarchical objective function. The solver produces provably optimal centralized solutions. Preliminary computational results show the efficiency of our approach.

Introduction

Distributed planning and scheduling problems arise in many contexts such as supply chain management, coordinating space missions, or configuring military scenarios. Such problems usually consist of several agents that need to perform tasks in order to achieve a common goal.

Depending on the application at hand, a distributed planning problem may be subject to several uncertainties: the actual outcome and duration of executing a task, changing environmental conditions, etcetera. Consequently, adaptive solution methods must be able to quickly compute possible alternatives in order to update the schedule, if necessary. In this work we present an efficient method to compute such (alternative) solutions.

We will focus on particularly difficult planning problems; those for which the common goal of the agents consists of a hierarchical objective function. Because these problems are not easily modeled and solved by traditional solvers for planning and scheduling, we propose to use *constraint programming* instead. Our approach can be used to (re-)compute individual agent's schedules, or to compute a solution from a centralized perspective. An important consequence of our approach is that it enables us to compute *provably optimal* solutions.

Problem Specification

The problems we consider are *task structures*, based on the C_TAEMS language (Boddy *et al.* 2005). Due to practical reasons we consider a restricted version of C_TAEMS in this work, which we will describe below. An example of such (restricted) C_TAEMS task structure, is presented in Figure 1.

A C_TAEMS task structure is a tree-like structure, composed of *tasks* and *methods*. Each method is owned by, and may be executed by, an agent. The resulting quality of execution contributes to the quality of its parent task by means of a *quality accumulation function* (QAF). The parent task in turn contributes again to the quality of its parent, all the way up to the root of the task structure (in this case TaskGroup1). There are four different QAFs: min, max, sum and sync_sum. The first three QAFs respectively denote the minimum, maximum and sum of the quality over the subtasks. The latter QAF, sync_sum, denotes the sum of the quality of all subtasks that start at the same time; it is a synchronized sum. The overall goal is to maximize the quality of the root task. Temporal restrictions on the execution of the tasks are imposed by release times and deadlines (indicated by a time window `TW`), and by *enabling* relations between tasks or methods (depicted as bold arcs). Finally, each agent is restricted to execute at most one method at a time. For a more detailed description we refer to (Boddy *et al.* 2005).

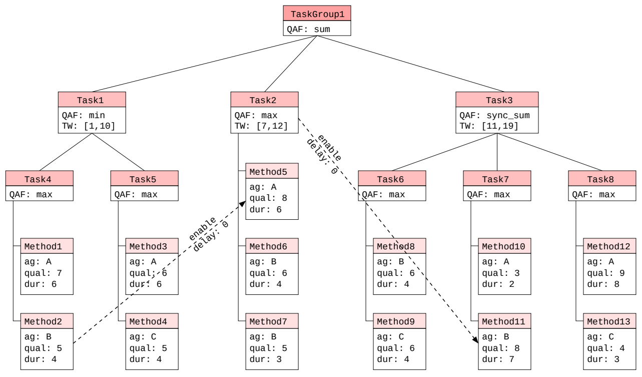

Figure 1: *Example of a C_TAEMS task structure. For each method,* `ag` *denotes its agent,* `qual` *its quality and* `dur` *its duration. Tasks may be subject to a time window* `TW` *and accumulate quality of their subtasks via a quality accumulation function* `QAF`.

Approach

Several candidate solution methods exist to compute a solution to (deterministic) distributed planning problems. For example, one can use a mixed integer programming solver, a satisfiability solver, or a planning solver. We have chosen to use *constraint programming* (CP) for the following reasons. The most important reason is that constraint programming has a rich modeling language which is very convenient to express the problem. Moreover, the underlying CP solver is relatively robust with respect to the addition of new constraints, and the search can be controlled entirely by the user.

In contrast, although the problem can be modeled to fit a mixed integer programming or satisfiability solver, we expect the respective solvers to suffer from the time granularity of the problem. This is not the case with a CP model. Finally, most planning solvers do not offer an easy way to model more complex objective functions.

We next give a brief overview of the core of our CP model. The key variables of our model are the variables that control the execution of the methods. They are represented by a variable $\texttt{method}_i$ for every task i that has methods as its leafs. The domain of this variable is given by its leafs (methods) and the no-execute option (denoted by method 0).[1] Hence, if $\texttt{method}_i = j$, then task i is executed using method j. To infer the agent, quality and duration of task i, we use so-called "element" constraints. Let the agent, quality and duration of a respective method j of task i be given by $ttag[i][j]$, $\texttt{qual}[i][j]$ and $\texttt{dur}[i][j]$. We introduce variables $\texttt{agent}_i$, $\texttt{quality}_i$ and $\texttt{duration}_i$ and impose the constraints:

$$\begin{aligned}
\texttt{agent}_i &= \texttt{ag}[i][\texttt{method}_i],\\
\texttt{quality}_i &= \texttt{qual}[i][\texttt{method}_i],\\
\texttt{duration}_i &= \texttt{dur}[i][\texttt{method}_i].
\end{aligned}$$

For the temporal relations we introduce variables $\texttt{start}_i$ and $\texttt{end}_i$ for each task i to indicate its start and end time. Then the release time and deadline constraints are easily modeled. The 'enabling' relationships need a bit more care, as they depend on the activation time of the task, i.e. the first moment it receives positive quality.

Although the above description is very brief, it reflects the core of our CP model. In fact, the search space is only defined by the variables `method` and `start`. By constraint propagation, all other variables are instantiated automatically once these variables are set.

Computational Results

As a case study we use problem instances from the COORDINATORs project, which are available from the authors on request. The instances represent real-life problem instances of medium to large size. The instances are typically designed for a dynamic multi-agent environment. It means that for most methods, the duration time and the quality are drawn from outcome distributions rather than being single values. Moreover, certain methods have a chance of failure. In our experiments we have replaced the outcome distributions by expected quality and maximum duration to make the problem deterministic. We further neglect methods that have a chance of failure. Our model can be easily adapted however, to change these settings.

[1]For simplicity we assume that each such task can execute at most one method. Task structures can be transformed to meet this assumption, however.

Table 1 presents computational results on a number of test problems. As constraint programming solver we have used ILOG Solver, version 6.0. The experiments were performed on a Pentium III 550MHz, 4GB RAM, with a time limit of 300 seconds per instance.

instance	**#tasks**	**#methods**	**#enables**	**optimum**	**time (s)**
CExample3_min	16	43	5	20	0.02
CExample3_max	16	43	5	60	0.05
CExample3_sum	16	43	5	138	0.08
ten1	116	160	21	$\geq$31.6	limit
test1a	21	42	5	33	0.02
test1	21	42	5	37	0.07
test2	30	21	15	45.8	0.03
test3	85	288	11	$\geq$74.4	limit
test4	104	89	25	95.4	0.44
testa	21	56	5	36.8	0.03
testb	21	42	5	36	0.04
testc	21	70	5	33.6	0.06
sgp-50920-213447	23	28	4	20	0.02
sgp-50920-213845	63	144	82	$\geq$30	limit
sgp-50920-214347	23	56	5	18	0.04
sgp-50920-214707	29	60	8	28	0.04
sgp-50920-214737	36	81	15	30	0.05
sgp-50920-214829	40	90	20	40	0.07
sgp-50920-214847	48	152	32	40	0.09
sgp-50920-215026	23	28	6	20	0.02
sgp-50920-215507	26	48	11	13.6	0.09
sgp-50920-215553	26	80	13	10	0.06
sgp-50920-221059	38	112	10	32	0.13
sgp-50920-221139	53	160	10	36	0.09

Table 1: *Computational results on a number of* C_TAEMS *task structures. All instances are solved to optimality, unless the time limit (300s) has been reached. Here 'problem' denotes the name of the instance, '#tasks', '#methods' and '#enables' denote respectively the number of tasks, methods and enabling relations of the instance, 'optimum' denotes the optimal quality found, or the best solution found in case the time limit is reached, and 'time' denotes the total solution time in seconds.*

Although most instances are solved very quickly, some instances were not solved to optimality within 300 seconds. This is probably due to the large number of methods present in these instances. As a result, the domains of the `method` variables are also large, which typically has a negative effect on the performance of a CP solver. On the other hand, some other instances with large domains are solved very quickly, which motivates the need for a closer investigation of the hardness profile of these problems.

Conclusion and Future Perspectives

We have shown that constraint programming can be a very efficient tool to model and solve distributed planning and scheduling problems with a hierarchical objective function. As our approach produces provably optimal (centralized) solutions, it will be particularly useful to evaluate the performance of problem-specific decentralized solvers. Furthermore, we intend to investigate the hardness profile of these problems in more detail, for which an efficient solver is indispensable.

References

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R.; Vincent, R.; Long, A.; and Kohout, B. 2005. C_TAEMS Language Specification — Version 1.03.