Reasoning About and Dynamically Posting n-ary Constraints in ADOPT

Federico Pecora[*], P. Jay Modi[†] and Paul Scerri[‡]

Abstract

This article describes an approach to solving distributed constraint optimization problems (DCOP) with n-ary constraints. A key instance of this problem is distributed resource-constrained task scheduling, in which limited resource capacities implicitly imply n-ary relations among the start-times of the tasks. We describe ADOPT-N, an extension of ADOPT [16], a recent successful algorithm for DCOP. ADOPT-N is an optimal asynchronous distributed n-ary constraint optimization algorithm in which specific agents are empowered with n-ary constraint evaluation capabilities. We show how the algorithm's correctness and optimality relies on (1) the choice of which agents to dedicate to constraint evaluation, and (2) an admissible (partial) variable ordering. Moreover, we demonstrate empirically how ADOPT-N's performance depends on how much knowledge about the n-ary violations that can arise during resolution can be provided to the algorithm.

1 Introduction

Solving constraint optimization problems involves computing an assignment of variables such that a global objective function is minimized (or maximized). The constraints correspond to value functions in the form $f : D_i \times D_j \rightarrow \mathbb{N}$, where D_i and D_j are the domains of the two variables involved in the constraint. For instance, $f_{\langle v_1, v_2 \rangle}(0,1) = 3$ asserts that the cost of the assignment $\langle x, y \rangle = \langle 0, 1 \rangle$ has a cost of 3 (a form of soft constraint.) This broad problem has been the focus of much research, and in particular poses interesting challenges related to the issue of solving distributed constraint optimization problems (DCOP) [1, 12, 23, 8, 6, 21, 10]. A DCOP differs from its non-distributed counterpart in that the computation is distributed among several agents, each reasoning upon a local constraint optimization problem.In the past years, a variety of efficient approaches for distributed constraint optimization have been studied [15, 11, 13, 19].

This article describes an approach to solving DCOPs with n-ary constraints, i.e., constraints with $n > 2$ variables of the form $f : D_1 \times \ldots \times D_n \rightarrow \mathbb{N}$. In this article we present ADOPT-N, an extension to ADOPT [16], which takes into account

[*]Institute for Cognitive Science and Technology, Italian National Research Council, federico.pecora@istc.cnr.it — Also affiliated with Dipartimento di Informatica e Sistemistica, University of Rome "La Sapienza"

[†]Department of Computer Science, Drexel University, Philadelphia PA, pmodi@cs.drexel.edu

[‡]Carnegie Mellon University, Robotics Institute, Pittsburgh PA, pscerri@cs.cmu.edu

n-ary constraints. The algorithm is an optimal asynchronous distributed constraint optimization algorithm which employs specialized agents to enforce n-ary constraints. The n-ary constraints may be known a-priori, or may become explicit only during problem resolution.

The ability to take into account constraints which are not known at the time of problem definition is fundamental in the context of many applications. In general, this is the case when n-ary relations among variables are intensional, and computing the extension is expensive and/or unnecessary. The term *constraint posting* refers to the capability of deducing and enforcing constraints that are not modeled in the problem during resolution. A good example of when constraints should be posted rather than modeled extensionally is distributed resource-constrained task scheduling (D-RCTS): a set of tasks to be allocated over time is given, as well as a set of (binary) constraints which define desired temporal relations among the tasks; in addition, all tasks consume a certain quantity of a given set of resources; the objective is to find an allocation in time of the tasks such that it never occurs that the resource requirements of the tasks exceed the capacity of the resources. Limited resource capacities implicitly imply n-ary relations among the start-times of the tasks.

A key feature of ADOPT-N is the ability to perform constraint posting. This capability is widely employed in specialized constraint reasoners such as schedulers[1]. Conversely, ADOPT-N is conceived to perform constraint posting while retaining the more general characteristic (along with its pros and cons) of distributed constraint optimization. As a consequence, the algorithm is "parametric" with respect to the particular domain-specific reasoning which is responsible for deducing the n-ary constraints to be posted.

ADOPT-N is correct and optimal due to (1) the choice of which agents to dedicate to constraint evaluation, and (2) an admissible (partial) variable ordering. This is true also for the constraint posting capability of the algorithm. In this article we demonstrate empirically how ADOPT-N's performance in the constraint posting setting depends on how much knowledge about the n-ary relations that can arise during resolution can be provided to the algorithm.

2 Resource-Constrained Task Scheduling

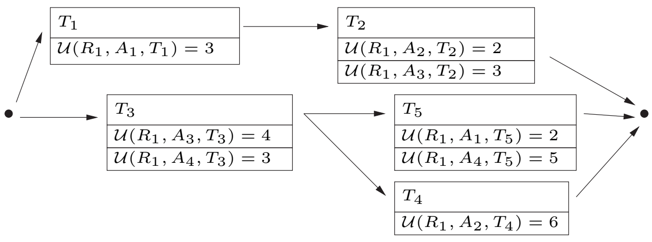
A domain in which constraint satisfaction/optimization problems are typically rich in n-ary constraints is resource-constrained scheduling. In this section we outline the specific category of scheduling problems which we use to describe and evaluate ADOPT-N throughout the paper.

The D-RCTS problem can be stated as follows. A set $\mathcal{A}$ of agents is responsible for carrying out a set of tasks $\mathcal{T}$. Every task involves one or more agents, and the participation of an agent A in task T requires the use of one or more given resources $R \in \mathcal{R}$ in the amount $\mathcal{U}(R, A, T)$. Each resource has a finite capacity $\mathcal{C}(R)$, and a set of precedence constraints $\mathcal{P}$ among the tasks is given. The precedence constraints are expressed in the form $T_i \prec T_j$, meaning that task T_i must occur before T_j. The

[1]For instance, non-distributed approaches to resource-constrained scheduling typically rely on binary constraint posting, i.e., precedence constraints are posted between pairs of tasks such that the excessive resource usage of the resources is gradually levelled (e.g., see [7].) One of the reasons for not posting n-ary constraints is that binary constraints can be propagated efficiently.

aim of the agents is to cooperatively devise an allocation of tasks in time which (a) is such that an agent only performs one task at a time, (b) satisfies the given precedence constraints, and (c) is such that the combined resource usage of the tasks is below the resource capacities at all times.

A D-RCTS problem can be visualized in the form of a precedence graph. The example in figure 1 depicts a problem with four agents and five tasks. The problem contains only one resource for simplicity (R_1) and the notation $\mathcal{U}(R_1, A_i, T_j) = n$ denotes that agent A_i requires n units of R_1 to perform task T_j.



T_1
$\mathcal{U}(R_1, A_1, T_1) = 3$
T_2
$\mathcal{U}(R_1, A_2, T_2) = 2$
$\mathcal{U}(R_1, A_3, T_2) = 3$
T_3
$\mathcal{U}(R_1, A_3, T_3) = 4$
$\mathcal{U}(R_1, A_4, T_3) = 3$
T_5
$\mathcal{U}(R_1, A_1, T_5) = 2$
$\mathcal{U}(R_1, A_4, T_5) = 5$
T_4
$\mathcal{U}(R_1, A_2, T_4) = 6$

Figure 1: A simple example consisting of five tasks, four agents and one resource. The edges represent precedence constraints between tasks.

Figure 2 shows the infinite capacity solution (schedule) to the problem (a), as well as a solution which maintains the profile of the resource under the capacity 11 of $R1$ (b). As is the case in solution (a), if the combined resource usage of the tasks in a schedule exceeds the capacity of a resource at time t we say that there is a resource peak at time t. Both solutions represent an allocation of tasks to time-points ($t = 0, t = 1, \ldots$) in

Time / Agent	$t = 0$	$t = 1$	$t = 2$
A_1	T_1	T_5	
A_2		T_2	T_4
A_3	T_3	T_2	
A_4	T_3	T_5	
Profile of R_1	10	**12**	6

(a)

$t = 0$	$t = 1$	$t = 2$	$t = 3$
T_1		T_5	
	T_4		T_2
T_3			T_2
T_3		T_5	
10	6	7	5

(b)

Figure 2: Two admissible allocations of tasks to time (schedules) for the example above: capacity $\mathcal{C}(R_1) = \infty$ (a), and $\mathcal{C}(R_1) = 11$ (b).

which mutex, agreement and precedence constraints are satisfied. These constraints state, respectively, that no agent can perform more than one task at a time, all agents involved in a given task have agreed to perform the task at the same time, and that the tasks are allocated on the time-line according to the given precedence constraints.

A D-RCTS problem can be formulated as a DCOP for resolution with ADOPT. Notice that the above problem is very similar to meeting scheduling [17], and as shown in [14], a number of different DCOP encodings can be employed. For the purposes of this paper, we employ the EAV (Events as Variables) formulation.

Research in scheduling with limited resource capacities [2, 5] has led to a number of (centralized) CSP-based scheduling techniques. Given the nature of these problems, which are characterized by the temporal constraints which define the precedence network on one hand, and the resource contention introduced by capacity limitations on the other, perhaps the most effective solution strategy has proved to be profile-based scheduling. In very few words, this resolution strategy is based on the observation that, in the example above for instance, it is sufficient to add the precedence constraint $T_2 \prec T_5$ in order to avoid the resource peak, a widely used technique known as precedence constraint posting [22, 9, 18, 7]. Notice, though, that this constraint does not necessarily avoid other resource peaks, such as the situation in which T_5 and T_4 occur concurrently, for which yet another precedence constraint ($T_4 \prec T_5$) should be posted. Overall, limited resource capacities entail n-ary relations among the start-times of the tasks. Moreover, these n-ary relations cannot be modeled explicitly in the problem definition, since deducing them requires to solve the (single-agent) scheduling problem in advance.

For this reason, we use D-RCTS as a running example and evaluation benchmark for ADOPT-N. The idea we pursue is thus to cast the D-RCTS problem as a DCOP (according to the EAV formulation) and to employ the constraint posting capabilities of ADOPT-N to minimize resource conflicts.

3 Overview of ADOPT

In DCOP each agent controls one or more variables. Constraints exist across agents, thus requiring coordination to find optimal solutions. ADOPT is optimal and asynchronous, but can be sped up while retaining quality guarantees [16].

The main idea behind ADOPT consists in allowing each agent to backtrack on a decision whenever it recognizes that another solution may be better. This behavior implements an “opportunistic” best-first search strategy: whenever an agent receives information indicating that a different assignment choice for its variables would lead to a lower cost, it backtracks on its decision. In other words, during the resolution process each agent continuously chooses the value assignment which improves the currently known *lower bound*. Notice that lower bounds can be computed without accumulating global cost information. This lower bound is iteratively refined as other agents communicate their own cost information.

For simplicity and without loss of generality, in the following description we assume that each agent A_k has only one variable (thus the terms agent and variable are equivalent). Each variable can be assigned values belonging to its domain $D_k = \{d_0, \ldots, d_n\}$.

The scheme of message passing in ADOPT is grounded on the definition of a partial order of priorities. Before the algorithm starts, the agents are organized in a Depth-First Search (DFS) tree. Each agent is assumed to know the tree before execution. Given the constraint graph which defines the DCOP, the tree is constructed in such a way that siblings are not bound by constraints, while constraints can be present between ancestors and descendants. More specifically, the priority tree is constructed iteratively from root to leaves according to an ordering heuristic, such as most-constrained-first (MCF) [4].

The ADOPT algorithm proceeds as follows. After an initialization step in which all agents select the value for their variable v_k which has the lowest cost, the agents enter a loop in which they send messages to and listen for messages from other agents. Specifically, parents send VALUE messages to their descendants. The descendants of a variable v_k are those variables with which v_k is involved in a constraint, and which are lower in the priority tree. The aim of VALUE messages issued by an agent v_k is to inform its descendants of the current choice of variable assignment val. Upon reception of a VALUE message, each agent computes the cost of its ancestor's choice and returns it to its parent by means of a COST message. Each agent maintains the lower bound cost of variable assignments $lb(v_k, d_i)$ by summing their costs to the costs reported by their children. In addition, a context field is attached to COST messages, containing the sender's view of higher agents' assignments at the time of cost calculation. Since calculated costs are dependent on the values of higher variables, this field is necessary because it allows an agent to reinitialize its costs when it receives a COST message containing a context which is incompatible with its current view.

Since the best-first search scheme allows agents to abandon potentially optimal solutions, ADOPT also implements an efficient mechanism for reconstructing these solutions when others appear to be sub-optimal. To this end, each agent maintains a backtrack threshold. The details of how this parameter is maintained are not described in the algorithm sketched above. For our purposes, it is sufficient to mention an agent's threshold represents an "allowance" on backtracking, thus an agent does not backtrack on its decision unless the computed lower bound of its current decision increases beyond the threshold. The modifications implemented in ADOPT-N do not affect termination, thus we do not dwell on further details of ADOPT, the details of which can be found in [16].

4 Dealing with n-ary Constraints

The naïve way to handle n-ary constraints is to "inject" messages that inform the agents controlling the variables involved of the additional cost of assignments which conflict with the n-ary constraint. Intuitively, agents would treat these messages like any other COST message and adjust values appropriately.

However, this naïve approach must deal with several issues. One issue which must be dealt with first of all is to identify which agents should be responsible for evaluating the n-ary constraints and generating these cost messages. Let us suppose that the problem contains an n-ary constraint on the values of variables in the set $V' \subseteq V$. Only an agent which has knowledge about the assignments of all variables in V' can be a candidate constraint evaluator. However, ADOPT specifically minimizes knowledge of other variable values, hence in general, no agent will be able to compute the cost of an n-ary constraint. For this reason, some of ADOPT-N's agents are augmented with the capability to perform the evaluation of n-ary constraints. We henceforth refer to these agents as n-agents. Their role is to assess the choices made by the other agents on the variables in V' in the light of the n-ary constraint. Given that n-agents must be provided with knowledge which in general no other agent in the problem possesses, we start by assuming that n-agents are additional, dedicated agents who do not participate in the decision making directly, rather whose role is solely to monitor the choices made by the other agents controlling relevant variables and injecting the cost of n-ary constraint infringing assignments into the message passing schema. As shown at the end of this section, this assumption can be relaxed, and n-agents will be designated among the original agents in the problem.

We augment the set of agents as it is defined in the initial problem formulation as follows. For each n-ary constraint on variables $V' \subseteq V$, an n-agent A' is included is placed in the hierarchy so as to be a descendant of all variables in V'. This is a necessary condition since the n-agent must receive VALUE messages from all relevant agents in order to gather a complete context. In addition, since the n-agents do not decide assignments, they will never have to send a VALUE message, rather they are solely responsible for sending COST messages. Overall, the n-agents send COST messages reporting the local cost of the assignment determined by their ancestors, and if the context indicates that the n-ary constraint they are responsible for is not violated, then a local cost of zero is reported to the parent variable(s). Conversely, if the n-ary constraint is violated, then the reported cost amounts to the cost of the n-ary constraint.

The natural place for n-agents in the priority tree is as children of the subtrees containing the variables in V'. As we show in the next paragraphs, the placement of the n-agents in the priority tree must be subject to further restrictions in order to ensure both correctness and optimality.

4.1 Optimal Strategies for n-ary Constraints

Since the variables in V' are placed in the DFS tree according to ADOPT's ordering heuristic, the naïve way to include n-agents (henceforth denoted by $v', v'', \ldots$) in the ordering is to append them to the tree as children of those paths from root to leaf which contain one or more variables in V' (see figure 3.) However, the naïve approach leads to two problems, namely double counting of costs and assignment masking.

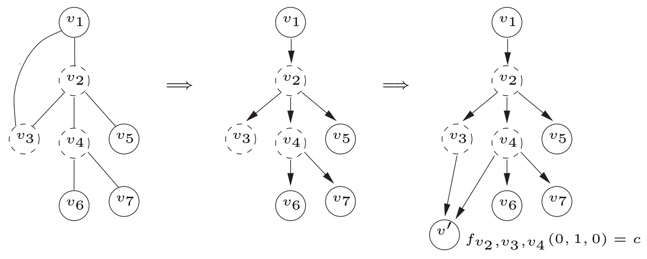


Figure 3: Variable ordering where an n-agent (v') has multiple parents: the binary constraints can be taken into account to generate the priority tree, and then v' it accommodated so as to terminate one path from root to leaf for every variable involved in the n-ary constraint (v_2, v_3 and v_4). However, this variable ordering is not admissible because it entails double counting of costs and assignment masking.

The first problem occurs because an agent which emanates a cost messages amounting to c towards two distinct branches will entail that the agent at the root of the subtree where the two branches originate perceives a cost of $2c$ associated to its choice (and the choices of its ancestors.) This is in contrast with the assumptions underlying the correctness of ADOPT (see [16].) One simple way to avoid this problem would be for the n-agent to only send the cost to one parent. Nonetheless, this would still incur in the following assignment masking problem.

Assignment masking occurs because variables pertaining to distinct subtrees are independent, i.e., it is never necessary for one to change its value assignment as a result of the value assignment chosen by the other variable. This property clearly does not hold in the case of n-ary constraints, the nature of which makes value assignments of variables in distinct subtrees dependant. Thus, multiple parents would entail a non-admissible pruning of the search space.

An example of assignment masking can be seen in the problem shown in figure 3. Let the current assignment be $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 0, 0, 1, 0, 0, 0, 0\rangle$. Since this conflicts with the n-ary constraint $f_{v_2,v_3,v_4}(0, 1, 0) = c$, the n-agent will emanate a COST message to v_3 and v_4. At this point, both these agents will associate the cost c to the assignment described in their context. Since, v_3's context does not contemplate the value assignment of v_4 and vice-versa, this lower bound will remain valid for every value assignment which differs from the above only for the value of v_3 (from v_4's perspective) and vice-versa. Supposing that the optimal solution to the DCOP is just one such assignment, e.g., $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 0, 0, 1, 1, 0, 0, 0\rangle$ (which does not violate the n-ary constraint), then this solution will be "masked" by v_3's lower bound, since v_3 may not, given this context, switch its value back to 1. On the contrary, if an optimal solution were $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 1, 0, 1, 1, 0, 0, 0\rangle$, then this solution would not be masked since the different value of v_1 (an ancestor of v_3) would change v_3's context and thus allow the re-initialization of its lower bound.

The considerations made above suggest that the key to maintaining optimality while allowing ADOPT-N to correctly reason upon valued n-ary constraints is to modify the ordering heuristic so as to ensure the local serialization of the variables involved in an n-ary constraint. More specifically, ADOPT-N implements the following locally-serializing meta-heuristic (see figure 4, top):

1. Recursively choose as next successor (root at first iteration) the variable returned by the chosen heuristic.

   1a. If the chosen variable is involved in an n-ary constraint, choose as next successor another variable involved in the same constraint and repeat point (1a) until all variables in the constraint have been placed.

   1b. Otherwise, if the chosen variable is not linked to its predecessor, branch off the lowest-priority already chosen variable that is, and repeat point (1).

3. For each n-ary constraint, append the n-agent as the successor of the lowest priority variable involved in the n-ary constraint.

The above locally-serializing procedure achieves the goal of avoiding double counting of costs and solution masking, thus ensuring correctness and optimality. Moreover, the locally-serializing procedure capitalizes on the problem partitioning by affecting the depth of the tree only as much as the n-ary constraint require. As shown in [16], the strategy of prioritizing variables according to trees that are as shallow as possible pays off in terms of computational overhead, since the assignments of variables in distinct subtrees do not affect each other. Nonetheless, notice that point (1a), which ensures the local serialization of variables involved in n-ary constraints, inevitably alters the order of the variables as it is determined by the ordering heuristic.

An alternative strategy is to forefit partial ordering in favor of maintaining the benefits of the ordering heuristic. A globally-serializing procedure works as follows: first, produce a DFS tree according to the heuristic of choice (e.g., MCF), and then serialize the tree to obtain a priority chain (i.e., all variables in the problem are on one path from root to leaf.) The n-agents can then be added to the chain as above (see figure 4, bottom.)

Overall, the locally-serializing strategy adopts a conservative approach to serialization, but it compromises the quality of the relative ordering between variables; conversely, the globally-serializing procedure completely forfeits the capability to minimize the interaction between agents during the solving process, but it maintains the ordering heuristic intact. The advantage of using one strategy rather than the other depends largely on the structural characteristics of the constraint graph. Specifically, a problem in which many variables are involved in n-ary constraints will more likely benefit from the globally-serializing procedure, while one in which few and low-arity n-ary constraints are present will lead to small amounts of local serialization, thus allowing ADOPT-N to take full advantage of the partial DFS tree ordering.

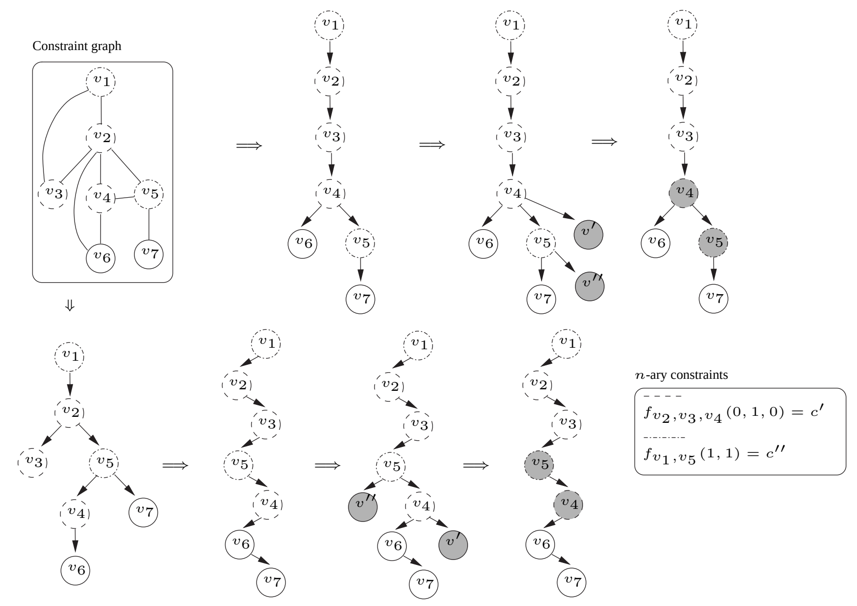


Figure 4: Priority tree ordering in ADOPT-N for a problem containing two n-ary constraints involving, respectively, $\{v_2, v_3, v_4\}$ and $\{v_1, v_5\}$. The locally-serialized strategy (top) serializes the DFS tree only as much as the n-ary constraints require, while global serialization yields a priority chain (bottom). Both strategies need not model a dedicated n-agent.

Since the constraint evaluating variables (v' and v'' in the example) are by definition always on a single path from root to leaf containing all the variables involved in the n-ary constraint, it is not necessary to model these variables as distinct from all other variables in the problem definition. Indeed, the n-ary constraint evaluating functionality implemented in a dedicated n-agent can be carried out by the agent involved in the n-ary constraint which has lowest priority (in the example, the functionality of v' and v'' is incorporated in v_4 and v_5.) Since the complexity of ADOPT is exponential in the number of variables, this is advantageous in terms of performance as it reduces the overhead for dealing with n-ary constraints to the computation of an additional component for the cost.

To conclude, we note that n-ary constraints have been used in [3] to enforce additional criteria in multi-criteria DCOP. The proposed mechanism is grounded on similar intuitions in that it too guarantees correctness and optimality by placing variables involved in n-ary relations on single paths from root to leaf. The principal differences with the present work are that the above cited work does not focus on n-ary constraints other than no-goods, and that ADOPT-N is conceived with the goal of n-ary constraint posting in mind.

5 Constraint Posting

The above variable ordering procedures rely on knowing all n-ary constraints a-priori. However, as motivated by the D-RCTS problem described in section 2, we wish to handle cases where such constraints arise dynamically during problem resolution. The one-path requirement must be upheld also in the event that the n-ary constraints are not known before hand but are deduced during problem resolution. To handle this, we must arrange the DFS tree to ensure that the constraints that are posted always involve variables which are on the same path from root to leaf. To this end, we employ the notion of *critical sets*, which represent groups of variables which together constitute the possible n-ary constraints that are to be posted.

In this section, we address the issue of understanding the impact of not knowing the n-ary constraints at the time of problem specification. Clearly, if we assume absolutely no knowledge on the n-ary relations which can arise, then the critical set will encompass all variables in the problem. Notice that this nullifies the advantage of using a locally-serializing strategy. Nonetheless, many domains do offer some insight as to where n-ary relations may arise. In the D-RCTS problem, where n-ary relations arise because of resource contention, it is possible to group a-priori a super-set of the variables involved in the peak.

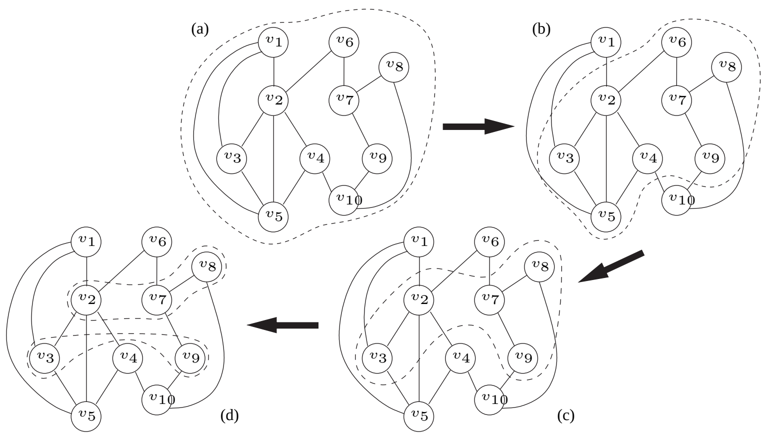


Figure 5: Situations (a), (b), (c) and (d) represent an increasing amount of knowledge on the n-ary relations that are to be posted.

In general, we can envisage more or less precise domain-specific strategies for determining the critical sets (see figure 5.) This reflects in how the n-agents are placed in the priority ordering, both in case of locally and globally-serializing orderings. Specifically, the more the critical sets single out precisely the n-ary relations that can occur, the more the locally-serializing ordering strategy can produce shallow DFS trees. Thus, the amount of knowledge given on the n-ary relations (i.e., the precision of the critical sets) affects the performance of the algorithm in that less knowledge tends to curtail the branching factor of the DFS tree, while more precise knowledge imposes less local serializations. Moreover, both in the case of local and global serialization, smaller critical sets will allow to designate as n-agents variables which are potentially higher in the hierarchy. The placement of n-agents higher in the priority tree entails that the costs incurred by n-ary constraint evaluation are propagated to fewer agents, thus resulting in better performance. The overall positive effect of critical sets which are as close as possible to being minimal is confirmed by the experimental evaluation in the following section, in which we compare two alternative strategies for building the critical sets, one in which domain knowledge is used more proficiently in order to minimize the size of the critical sets, and one in which the critical sets are built more "coarsely".

5.1 Constraint Posting in D-RCTS: Experiments

The D-RCTS problem benchmark on which the following experiments were carried out is drawn from the RCPSP/max problem set [20] (Resource Constrained Project Scheduling Problems, single mode project duration.) The J10 problem set, which was employed to obtain the instances of the RCTS benchmark, consist of project scheduling problems with ten activities, each with integer duration, a set of minimum time lags, a set of maximum time lags, and a set of capacity-bound resources. Each activity requires one or more resources for execution. A minimum time lag of t between two activities expresses the fact that one activity must begin at least t time units after the other, while a maximum time lag represents the fact that an activity must begin at most t time units after the beginning of the other.

The J10 RCPSP/max instances thus model a more general category of scheduling problems than simple (non-distributed) RCTS. In fact, the D-RCTS problems were obtained in two steps. First, an RCTS benchmark was obtained by ignoring the duration and maximum time lag information in the RCPSP/max instances. Second, agent-specific attributes were added to obtain D-RCTS problems. A random number of agents was assigned randomly to the tasks such that each task was executed by at least one agent.

Since our aim is to post n-ary constraints which avert resource contention, the critical sets represent a collection of variables whose concurrent assignment to the same value (i.e., tasks that occur at the same time) can potentially entail a resource peak. Notice that the size of a critical set is an upper bound on the arity of the n-ary relation, since it groups those variables which together *may* entail produce a resource peak. As a consequence, the evaluating agents (one for every critical set V_i) will post n-ary constraints whose arity is *at most* $|V_i|$.

The critical sets in general model a collection of constraints represented intensionally, and their evaluation occurs by means of a collateral, domain-specific analysis of the current assignment. The domain-specific behavior of ADOPT-N's evaluating agents is adapted to perform the evaluation of variable assignments on the basis of the resource-related information in the problem definition (i.e., $\mathcal{R}$, $\mathcal{C}$ and $\mathcal{U}$.) More specifically, at each iteration, the evaluating agents (one for each critical set) calculate the resource usage profile of the (partial) assignment they see in their context. If the combined usage of a resource on behalf of variables which are assigned to the same value (tasks which occur in the same time slot) exceeds its capacity, then the cost reported by the evaluating agent to its parent amounts to the amount of excess usage of the resource; otherwise, the normal cost of the assignment is reported (i.e., no n-ary constraint is violated, and the cost is computed normally.) This mechanism ensures that an optimal solution will be one in which the amount of excess resource usage (if any) is minimized. In the benchmark employed for the experiments, there exists for all problems at least one resource-feasible solution (i.e., no excess resource usage), thus the cost of the optimal solution is always zero.

The critical sets were defined through the following two alternative strategies. Specifically, the two strategies differ in how close the critical sets are to being minimal:

Same Resource (SR). The SR strategy consists in creating one critical set for every resource containing all variables which use that resource. Notice that the critical sets can overlap since tasks can use more than one resource. This strategy yields a very coarse characterization of the n-ary relations, since it characterizes as critical all groups of variables which use the same resource, disregarding completely the presence of the precedence constraints.

Resource Peak Analysis (RPA). RPA allows to evaluate the resource usage parameters in the light of the precedence graph. In practice, the transitive closure of the precedence graph is computed, from which all sets of tasks which can potentially occur together are extracted. Specifically, the tasks belonging to each critical set satisfy the following conditions: (a) the tasks are not connected in the transitive closure of the precedence graph, and (b) all tasks use the same resource.

The RPA strategy for determining the critical sets is clearly more precise than the SR strategy, since it curtails the size of the critical sets by taking into consideration also the precedence constraints in the D-RCTS problem. This is confirmed by the average number and arity of the n-ary relations (i.e., the number of critical sets and their size) determined by employing the two strategies on the 270 D-RCTS problems in the benchmark. Specifically, for SR the average number of constraints is 4.92 and the average arity is 9.79, versus 3.64 and 8.3 for RPA.

The 270 D-RCTS problem instances were solved in the following three settings. First, a relaxed version of the D-RCTS problems was solved. These problems consist in the DCOP formulations of the J10 problems ignoring the resource-related information ($\mathcal{R}$, $\mathcal{C}$ and $\mathcal{U}$), and thus are characterized by only agreement, precedence and mutex constraints[2]. We henceforth refer to these problems as *base problems*. Second, the complete D-RCTS problems were solved using ADOPT-N and the critical set generating strategy SR. Third, the complete D-RCTS problems were solved using ADOPT-N and the critical set generating strategy RPA.

Figure 6 shows the percentage of solved problems and the average solving time of the three solver settings. As demonstrated by the lower number of solved problems and the higher solving times, the resource constrained variants of the base problems constitute more difficult instances. Clearly, this is due to the fact that resource capacity limitations invalidate the assignments which would constitute an optimal solution in the base problem. Indeed, the optimal solutions for the resource constrained problems often correspond to sub-optimal assignments for the base problem, which are though

[2]Notice that using ADOPT-N with a locally-serialized ordering and no n-ary constraints or critical sets is equivalent to the original ADOPT algorithm.

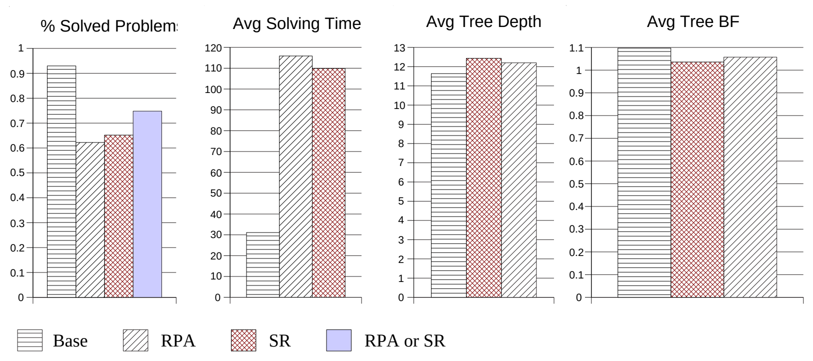


Figure 6: Fltr: average number of solved problems, average solving time [sec], average DFS tree depth, and average DFS tree branching factor.

optimal in the resource constrained setting because the capacity limitations invalidate all solutions which would otherwise be optimal.

While the higher complexity of resource constrained problems does not come as a surprise, it is interesting to analyze the performance of ADOPT-N given the two different critical set generating procedures. As shown in figure 6, the discrepancy between RPA and SR does not follow the intuition that a more profitable procedure for collecting the critical sets pays off in terms of performance. Indeed, the less sophisticated SR strategy seems to have a slight advantage. Nevertheless, the reason for this behavior can be found in the observation made earlier, namely that a gain in tree depth at the cost of subverting the ordering heuristic can be counter-productive. In fact, if the gain in branching factor of the problems obtained with the RPA strategy is not particularly strong (6, right). This points to the fact that the predominant factor which affects performance is not the degree to which the critical set generating procedure can maintain the DFS tree shallow, rather the specific ordering heuristic which is employed.

In order to compare the RPA and SR critical set generating strategies on the same variable ordering, we also ran the following experiments with the globally-serializing ordering strategy. These experiments were run on 405 D-RCTS problems, divided into three groups: $\mathcal{P}_{200}$, $\mathcal{P}_{100}$, and $\mathcal{P}_0$. Every group consisted in 135 problems, and each problem $p_i \in \mathcal{P}_{200} (\in \mathcal{P}_{100})$ was obtained from problem $p_i \in \mathcal{P}_0$ by increasing the capacity of all resources by 200% (100%). Each problem was solved by ADOPT-N using the globally-serializing strategy with the MCF ordering heuristic. The aim of these three groups was to ensure that the results (i.e., the relative advantage of RPA or SR) would depend solely on the placement of the n-agents in the priority chain, and not on the level of resource constrainedness of the problem.

The results (see figure 7) show that the more informed RPA strategy, which makes a more profitable use of domain information, performs 15.9%, 16%, and 9.2% better than the less informed SR strategy in the three cases. These results show that RPA scales better than SR. This is because RPA identifies more precisely the structure of the n-ary relations, in that it approximates their size and arity better by taking into account also the precedence constraints. In fact, the average number and size of the critical sets (i.e., the envelope of the implicit n-ary relations) determined by the two strategies is 4.92 and 9.79 for SR, versus 3.64 and 8.3 for RPA. The experiments thus substantiate the

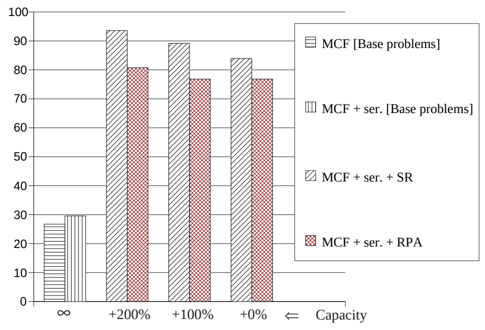


Figure 7: Average solving time [sec] with SR and RPA strategies and the globally-serializing strategy, i.e., which does not alter the MCF heuristic used to compute the variable ordering.

observation made earlier, namely that the placement of n-agents at higher levels within the priority ordering can strongly affect the performance of the algorithm.

The specific placement of the n-agents within the priority chain can be analyzed through the the parameter $\pi = \sum_i \frac{\rho(i)}{l}$, where l is the length of the chain and $\rho(i) = i$ if the agent at level i of the chain is an n-agent, 0 if it is not. This parameter measures how sparse the n-agents are within the chain, thus high values indicate that the n-agents are concentrated at the bottom of the chain. In the present benchmark, for 75% of the instances in which RPA performed better than SR, π was higher for SR than RPA, thus corroborating the fact that agent placement strongly affects performance.

Notice, finally, that the increasingly tight resource constraints do not worsen performance in the general case. In fact, tight capacity constraints often have the effect of pruning large portions of the search space, thus invalidating many of the assignments worth exploring for the underlying constraint optimization problem.

6 Conclusions and Future Work

In this paper we have dealt with two aspects related to reasoning about n-ary constraints in ADOPT. First, we have detailed the enhancements necessary to deal with n-ary constraints that are specified in the problem definition. This is achieved in ADOPT-N, a variant of the original algorithm which maintains optimality guarantees by means of a variable ordering heuristic which produces locally or globally serialized DFS trees. We have shown, both in theory and empirically, that there is a tradeoff between local and global serialization. The former strived to maintain shallow DFS trees, but at the cost of subverting the variable ordering heuristic, while the latter preserves the benefits of the ordering heuristic at the cost of forfeiting partial ordering between variables. The relative advantage of one method or the other are largely dependant on the domain. Our experimental evaluation tips the scale in favor of global serialization in the D-RCTS domain.

Secondly, this paper tackles the problem of taking into account constraints which become known only during problem resolution, i.e., which are either deduced as a result of a collateral domain-based computation, or which are communicated to the agents from an external source. This capability is fundamental for application domains in which n-ary constraints among variables are intensional, and computing the extension is expensive and/or unnecessary. More specifically, we show how the impact of a-priori knowledge regarding the n-ary relations which can arise during resolution affects performance.

The problem instances constituting the D-RCTS benchmark are adapted from the single mode project duration RCPSP/max benchmark [20], which is widely used in the scheduling community. The experiments are aimed at assessing (1) the (expected) increase in difficulty of the resource-constrained problems (for which n-ary constraint posting is necessary) with respect to the base problem (containing only statically defined binary constraints), and (2) the influence of different amounts of a-priori knowledge on the n-ary relations which must be posted during resolution. With respect to the second point, our analysis reveals two insights. First, the performance of ADOPT-N on the D-RCTS benchmark is strongly affected by the ordering heuristic, thus the locally-serializing ordering strategy, which tends to subvert this heuristic, is not suitable for the D-RCTS benchmark. The second insight is that the more sophisticated RPA strategy for collecting critical sets entails a performance advantage, since it determines a more precise placement of the n-agents. This is thanks to the more precise way in which RPA isolates groups of variables which can contribute to resource contention, minimizing the size of these n-ary relations.

Future work will focus on providing further insight on implicit and explicit n-ary constraint reasoning in ADOPT-N. From the explicit point of view, we are interested in performing a more in-depth evaluation of the performance of ADOPT-N on problems with n-ary constraints in known domains, such as graph-coloring, as well as more application oriented domains, e.g., agent coordination in logistics problems. In the context of implicit constraint posting, we will focus on domains in which more precise knowledge on the n-ary relations entails an improvement in ADOPT-N's performance. More specifically, the present work has given us some insight as to why performance can benefit from more knowledge.

Acknowledgements

Federico Pecora is partially supported by MIUR (Italian Ministry of Education, University and Research) under project ROBOCARE: "A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components", L. 449/97 (`http://robocare.istc.cnr.it`). The first Author would also like to express his gratitude to Prof. Steve Smith and the Intelligent Coordination and Logistics Laboratory at Carnegie Mellon University, for having hosted him as a visiting scholar in the period June-December 2005, as well as Prof. Daniele Nardi and Dr. Alessandro Farinelli for having fostered interesting discussions on topics related to Multi-Agent Coordination.

References

[1] A. Barrett. Autonomy Architectures for a Constellation of Spacecraft. In *Proc. of the 5th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-99) Noordwijk, The Netherlands*, pages 291–296, 1999.

[2] M. Bartusch, R. H. Mohring, and F. J. Radermacher. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research*, 16:201–240, 1988.

[3] E. Bowring, M. Tambe, and M. Yokoo. Distributed Multi-Criteria Coordination in Multi-Agent Systems. In *Proceedings of Workshop on Declarative Agent Languages and Technologies at the Fourth International Joint Conference on Agents and Multiagent Systems (DALT-05), Utrecht, Netherlands*, 2005.

[4] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[5] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research*, 1998.

[6] R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[7] A. Cesta, A. Oddi, and S.F. Smith. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–135, 2002.

[8] H. Chalupsky, Y. Gil, C. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. Russ, and M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *International Conference on Innovative Applications of AI*, pages 51–58, 2001.

[9] C. Cheng and Stephen Smith. Generating feasible schedules under complex metric constraints. In *Proceedings 12th National Conference on Artificial Intelligence*, August 1994.

[10] Christian Frei and Boi Faltings. Resource allocation and constraint satisfaction techniques. In *Principles and Practice of Constraint Programming*, pages 204–218, 1999.

[11] K. Hirayama and M. Yokoo. An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proceedings of International Conference on Multi-Agent Systems*, 2000.

[12] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsushi Shinjoh, and Susumu Shimada. Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume VI, pages 739–743, Tokyo, October 1999.

[13] JyiShane Liu and Katia P. Sycara. Exploiting problem structure for distributed constraint optimization. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 246–254, San Francisco, CA, 1995. MIT Press.

[14] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS'04)*, pages 310–317, 2004.

[15] Roger Mailler and Victor Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society, 2004.

[16] P.J. Modi, W.M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[17] P.J. Modi and M. Veloso. Multiagent meeting scheduling with rescheduling. In *Distributed Constraint Reasoning, (DCR) 2004*, 2004.

[18] A. Oddi and Stephen Smith. Stochastic procedures for generating feasible schedules. In *Proceedings 14th National Conference on Artificial Intelligence*, July 1997.

[19] H. Parunak, A. Ward, M. Fleischer, J. Sauter, and T. Chang. Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 93–99, 1997.

[20] Christoph Schwindt. University Karlsruhe (TH), Institute for Economic Theory and Operations Research – Project Generator ProGen/max and PSP/max-library Website: http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/index_html, Accessed Jan. 2006.

[21] W.M. Shen and M. Yim. Self-reconfigurable Robots. *IEEE Transactions on Mechatronics*, 7, 2002.

[22] Stephen Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings 11th National Conference on Artificial Intelligence*, July 1993.

[23] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124, 1997.