

Coordination of Agent Schedules using Distributed Neighbor Exchange

Number 964

Distributed Search/CSP/Optimization ** Multiagent Systems (General/other)

* Scheduling ** Coordination And Collaboration

Abstract

Coordination of agent schedules in non-deterministic, distributed environments is computationally difficult. Distributed Constraint Optimization (DCOP) provides a rich framework for modeling such multi-agent coordination problems, but existing problem domains for DCOP focus on small (<100 variables), deterministic domains. We present a complete mapping to DCOP for large-scale team coordination problems that were used in the DARPA Coordinators program.

This domain requires distributed, scalable algorithms to meet difficult bounds on computation and communication time. To achieve this goal, we develop a new distributed neighbor exchange algorithm for DCOPs that scales to problems involving hundreds of variables and constraints and offers faster convergence to high quality solutions than existing DCOP algorithms. In addition, our complete solution includes new techniques for dynamic distributed constraint optimization and uncertainty in constraint processing. We show that our solution is very competitive with the general approaches used in the DARPA Coordinators program.

1 Introduction

Distributed Constraint Optimization (DCOP) is a general problem representation for multi-agent systems. Recent advances in DCOP algorithm development have led to an increasing number of application domains and focus on DCOP techniques. Recent applications of DCOP to real-world problems include sensor networks [Modi *et al.*, 2005], traffic flow cooperation [Junges and Bazzan, 2008], and event scheduling [Maheswaran *et al.*, 2004]. These existing problem domains for DCOP focus on small (<100 variables), deterministic domains. We present a mapping to DCOP for large-scale team coordination problems that were used in the DARPA Coordinators program.

While our mapping is generic and can be used with existing DCOP algorithms, the Coordinators problem domain requires distributed, scalable algorithms to meet difficult bounds on

computation and communication time. To achieve this goal, we develop a new DCOP algorithm that processes utility distributions and scales to problems involving hundreds of variables and constraints. We show that our algorithm outperforms other DCOP algorithms for this domain and that our approach is competitive with other general approaches used in the DARPA Coordinators program.

Existing DCOP formalizations also require deterministic utility outcomes for constraint assignments. This representation precludes reasoning about uncertainties within a DCOP algorithm. We introduce a new DCOP formalization that allows for non-deterministic constraint functions that evaluate to discrete utility distributions. These discrete utility distributions take the place of integer values in DCOP algorithms and allow the processing of non-linear functions over combined distributions. This extension allows our algorithm to incorporate a risk avoidance strategy to deal with uncertainties that are part of the Coordinators problems.

We begin with an introduction of the CTÆMS problem representation and the existing DCOP formalism. We then extend the existing DCOP formalization to include constraint functions with discrete utility distributions. We detail the mapping between a CTÆMS problem representation and the extended DCOP formalism. We introduce a new neighborhood exchange algorithm for DCOP that scales to problems involving hundreds of variables and constraints. We show that our solution is very competitive with the general approaches used in the DARPA Coordinators program.

2 C-TÆMS coordination problem

Multi-agent task planning and scheduling problems require a rich language for domain representation. The original TÆMS (Task Analysis, Environment Modeling, and Simulation) language was developed to provide a domain independent, quantitative representation of the complex coordination problem [Horling *et al.*, 1999]. A C-TÆMS problem instance contains a set of agents and a hierarchically decomposed task structure. Nodes in the graph are either complex tasks (internal nodes) or primitive methods (leaf nodes). Each node may have temporal constraints on the earliest start time and the deadline. Nodes may also have non-local effect (NLE) constraints that represent hard (enables and disables) and soft (facilitates and hinders) node relationships. Methods have probabilistic outcomes for duration, quality, and cost. Tasks

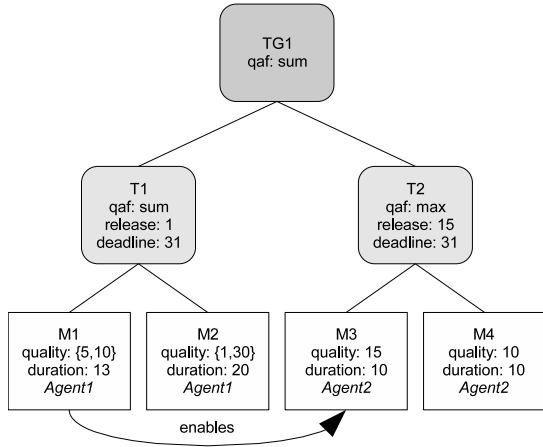


Figure 1: An example C-TÆMS problem instance.

have a quality accumulation function (QAF) that describes how quality accrues at the task based on the quality of its sub-tasks and methods. Some basic QAFs include sum, sumand, syncsum, min, max, and exactlyone. In the sample C-TÆMS problem instance in Figure 1, the node T1 represents a task with M1 and M2 representing a decomposition of this task into submethods. T1 has constraints for earliest start time of 1 and deadline of 31. The accumulated quality at T1 is a sum of the qualities of the executed submethods.

2.1 Existing Approaches

Three teams participated in the second phase of the Coordinators project and used very different approaches. Detailed descriptions of the approaches can be found in [Maheswaran *et al.*, 2008; Witwicki and Durfee, 2007; Smith *et al.*, 2007]. The best performing team used a risk avoidance strategy based on predictability and criticality metrics (PCM). These metrics did not aim to optimize an approximate global utility function, but instead minimized chances for conflicts while opportunistically inserting changes that did not negatively impact the current schedule. A second approach used simple temporal networks to create a flexible time schedule of methods that changes over time using constraint propagation (FTS). Agents choose to modify the schedule when speculation indicates local utility gain is greater than neighboring utility loss. The third approach used distributed MDPs to approximate optimal execution policies (MDP).

Our approach is most similar to the FTS approach because we will use DCOP to speculatively optimize over a flexible execution schedule. However, we design the constraints with discrete value distributions and reason over them using a risk aversion function. This function provides a similar role to the metrics used in the PCM approach as it is not a direct approximation of the global utility function.

3 Existing DCOP Formalization

DCOP has been formalized in slightly different ways in recent literature, so we will adopt the definition as presented in [Petcu and Faltings, 2005]. A Distributed Constraint Opti-

mization Problem with n nodes and m constraints consists of the tuple $\langle X, D, U \rangle$ where:

- $X = \{x_1, \dots, x_n\}$ is a set of variables, each one assigned to a unique agent
- $D = \{d_1, \dots, d_n\}$ is a set of finite domains for each variable
- $U = \{u_1, \dots, u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility for each combination of values among these variables

An optimal solution to a DCOP instance consists of an assignment of values in D to X such that the sum of utilities in U is maximal. Problem domains that require minimum cost instead of maximum utility can map costs into negative utilities. The utility functions represent soft constraints but can also represent hard constraints by using arbitrarily large negative values.

4 DCOP with Utility Distributions

We can extend the DCOP problem formalization to include uncertainty by allowing constraint evaluation functions to return a distribution instead of a single value. A global optimum is now an optimal distribution instead of a maximum (or minimum) sum. To evaluate the optimality of a distribution, evaluation criteria must be formalized. The optimal evaluation function may not be the same for all problems for all agents. Thus we must include the evaluation function as part of the extended DCOP problem. We extend our previous DCOP formalization for this:

- $U = \{u_1, \dots, u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility distribution for each combination of values among these variables
- $u = \{(u_p^1, u_v^1), \dots, (u_p^t, u_v^t)\}$ is a distribution of probabilities and values such that $\sum_{r=1}^t u_p^r = 1$
- $E = \{e_1, \dots, e_n\}$ is a set of evaluation functions for each variable that reduce a utility distribution to a single utility value; $e(u) = v$ where v is a single utility value

This extension requires extra computation and memory proportional to the maximum allowed size of a distribution. However, it allows processing of non-linear functions over combinations of utility distributions. Most DCOP algorithms are based on sum and max functions, which are easily extendable to discrete distributions. Evaluation functions can include calculations for the median and for risk assessment for both risk seeking and risk averse behavior. In our work we use a risk aversion function that calculates a new weighting for the sorted values in the discrete distribution using integration over the function $2 * (1 - x)^2$. This function emphasizes worst case scenario utility for our Coordinators agent, avoiding volatile methods that may perform much worse than expected.

5 C-TÆMS mapping

We can observe that DCOPs naturally optimize global sums of utility, so mapping a C-TÆMS scheduling problem to a

DCOP appears easy at first. A naive approach could equate the quality produced by executing methods with utility and optimize the global sum given a set of constraints. However, the introduction of task interaction dependencies (non-local effects, NLE) and non-linear quality accumulation functions makes the job much more difficult. In order to correctly map the quality of the schedule to a global utility function we need to incorporate the interaction of QAFs, NLEs, and non-determinism.

5.1 Existing mapping

A mapping for a subset of C-TÆMS to DCOP is proposed in [Sultanik *et al.*, 2007]. The mapping using our formalization is:

- X = Each method is assigned to a unique variable.
- D = Unique domains for each variable containing all possible start times for the method assigned to the variable.
- U = Three types of utility functions:
 - Mutex constraints on all pairs of methods that share the same agent
 - For an NLE between two nodes, N_1 and N_2 , all methods in the subtree of N_1 have a precedent constraint with all methods in the subtree of N_2
 - Unary soft constraints on each method that apply a cost if the method is not scheduled

While this mapping is a good start, it is severely limited because it only allows one sum, min, and max QAFs, only one type of QAF per structure (no mixing sum with max QAFs), and only enables NLEs. It also requires deterministic task outcomes, so it cannot benefit from integrated risk analysis functions.

An extension to this mapping added syncsum QAF mappings and provided for multiple QAF types in the same structure without using n -ary constraints [Atlas and Decker, 2007]. We propose a modification to this extension that includes a full set of QAFs and NLE functions used in the DARPA Coordinators project.

5.2 Proposed mapping

Our proposed mapping for C-TÆMS to DCOP can be broken into two distinct parts: variable and constraint mappings. In the variable mappings we describe the domain values for the variable. For the constraints we present the utility function rules for involved variable values. All constraints are binary constraints in this mapping and produce discrete utility distributions as presented in Section 4.

5.3 Variables

Variables are created for each method and task in the C-TÆMS problem. In addition, a special end-time variable is created for each task with an outgoing NLE at or above it in the structure.

Methods

Method variables are created with all possible start times as values and an additional value for *not scheduled*. Additional values are created for each permutation of a modifier that affects the method, including a synchronization point and all incoming NLEs.

Tasks

Task variables can have several different sets of values depending on the type of QAF assigned to the task. These values describe how quality will accumulate from subtasks and methods. All task variables contain values for *no execution* and *execution allowed no quality* that force children not to execute or to not accumulate quality (respectively). Max, min, and exactlyone QAFs contain values representing the children that will accumulate quality. Sum QAFs and sumand QAFs have a single sum quality value. Sync sum QAFs contain values for all possible synchronized start times for descendant methods. All task domains with a sumand QAF ancestor also have a modifier flag to force execution if set to a quality accumulating value.

NLEs

Non-local effects do not have their own variables. However, if a non-local effect originates at a task, then a special task end time variable is created for the task. This task end time variable contains all possible ending times for descendant methods and a value for *not scheduled*.

5.4 Constraints

Constraints are created between each related node in the problem structure. There are four types of relationships we create constraints for: task-subtask, task-method, method-method at the same agent, and to and from nodes in a NLE. In addition special constraints are created for synchronization points. The actual implementation values for constraints will be problem specific, with the following defined values: Q_{max} is an upper bound for the maximum quality of the entire problem structure and Q_{M1} is the quality distribution for method $M1$. All hard constraints are enforced using $-Q_{max}$ as the utility for a constraint violation.

Task-Subtask

A task-subtask constraint enforces a correct accumulation of quality up from the methods through each of the QAFs to the root. Generally, the constraint enforces that a subtask may only accumulate quality if the task allows it. Additionally, this constraint propagates information about the forced execution modifier flags so that a subtask knows if the task is relying on it to produce some quality. If a task is assigned to accumulate quality, the implementation depends on the type of QAF:

- *max* - only the selected subtask may accumulate quality. All other subtasks must be set to *execution allowed no quality*.
- *sum* - all subtasks may accumulate quality.
- *min* - only the minimum valued subtask may accumulate quality. All other subtasks must be set to force execution with no quality.

- *syncsum* - all subtasks may accumulate quality.
- *exactlyone* - only the selected subtask may accumulate quality. All other subtasks must be set to *no execution*.
- *sumand* - all subtasks must be assigned a forced execution modifier flag.

Task-Method

A task-method constraint accumulates the quality for properly scheduled methods. Thus for a valid scheduled method, this constraint returns a value of Q_M . The value for Q_M includes any modification indicated by incoming NLE flags (for example it will be twice the original method quality if a facilitation flag is set with a factor 2). A constraint violation occurs if any method is not scheduled and is in the set of methods selected for scheduling by the task value. Again, the implementation for accumulating quality depends on the type of QAF:

- *max* - all scheduled, selected methods return $\max_{M_k \in T_M} \frac{Q_{M_k}}{|T_M|}$. All others return zero utility.
- *sum* - all scheduled methods return Q_M .
- *min* - all scheduled methods return $\min_{M_k \in T_M} \frac{Q_{M_k}}{|T_M|}$. Any unscheduled method is a constraint violation.
- *syncsum* - all scheduled methods return Q_M if start time equals synchronization point. All others return zero.
- *exactlyone* - only the selected method returns Q_M . Any other scheduled method is a constraint violation.
- *sumand* - all scheduled methods return Q_M . Any unscheduled method is a constraint violation.

An additional special task-method constraint is created from a syncsum task to all descendant methods. This is the only task-method constraint between nodes that are not directly connected in the problem structure. If the method has its synchronization modifier enabled but its start time does not equal the synchronization point, $-Q_M$ is returned for the utility.

Method-Method

A method-method constraint ensures that an agent is not scheduled to execute two methods at the same time. A mutex constraint is created between all pairs of methods at an agent. We prune all mutex constraints that have no possible overlap. For assignments that would cause overlap, the utility returned is equal to $\max_{k \in C_M} -Q_{M_k}$ where C_M is the set of methods involved in the constraint.

Non-Local Effects

A non-local effect (NLE) constraint may occur between tasks, methods, or both. In keeping with the CTÆMS specification, we decompose non-local effect constraints into only task-method and method-method relationships. A task-task constraint between T_1 and T_2 by definition behaves the same way as if the constraint existed from T_1 to all descendant methods of T_2 . For task-method NLEs we use the special end time variable to calculate the time at which the NLE is active. For method-method NLEs we use the start time plus the duration (which is a distribution) to determine when the NLE is active.

For each NLE type we return a value based on whether the NLE is active:

- *Enables* - if enable is not active at method start time, return $-Q_{max}$ if method has a sumand ancestor and $-Q_M$ otherwise.
- *Disables* - if disable is active at method start time, return $-Q_{max}$ if method has a sumand ancestor and $-Q_M$ otherwise.
- *Facilitate* - if facilitate is not active and method modifier flag is set, return $Q_M - facilitated(Q_M)$.
- *Hinders* - if hinders is active and method modifier flag is not set, return $hindered(Q_M) - Q_M$.

The CTÆMS specification also allows for proportionally active NLEs depending on how much quality has accumulated at a task that is an NLE source. We do not specifically map the proportional effect, but have constructed soft constraints in such a way that they encourage facilitation or discourage hindrance effects.

6 Distributed Neighbor Exchange Algorithm

We now introduce a new algorithm to solve large-scale distributed constraint optimization problems, the Distributed Neighbor Exchange Algorithm (DNEA). We developed this algorithm after finding that existing DCOP algorithms that can actually run on these problem sizes took too many message passing cycles to converge to solutions in our domain. DNEA is a local neighborhood search algorithm that exchanges potential gains for multiple assignments to all neighbors in each cycle.

6.1 Algorithm Phases

Our algorithm is similar in phases to other local value exchange based algorithms, including MGM, SCA, DBA, and max-sum[Bowring *et al.*, 2008; Farinelli *et al.*, 2008]. These algorithms exchange current variable assignments with neighbors, compute a maximization function based on neighboring assignments, and then choose to update the local variable assignment. There are two phases to our algorithm, value exchange and neighborhood utility exchange. An example of a simple execution path is shown in Figure 2.

Value Exchange

For each variable X with domain D_X in the DCOP instance, an associated agent chooses a value to assign the variable from a set of neighborhood utility valuations. At the beginning these valuations are zero, so the agent randomly assigns a value to X . On subsequent iterations, the agent will have received a utility exchange message for each variable that is a neighbor in the constraint graph. Each utility exchange message includes a utility value for each possible assignment in D_X for X . Each of these values represents the best local utility the neighboring variable can achieve for each possible value of X . The agent then sums all the neighbor valuations together and adds any local valuation X has for each assignment. The agent then chooses with probability p to change the value of X to the maximum assignment in this sum.

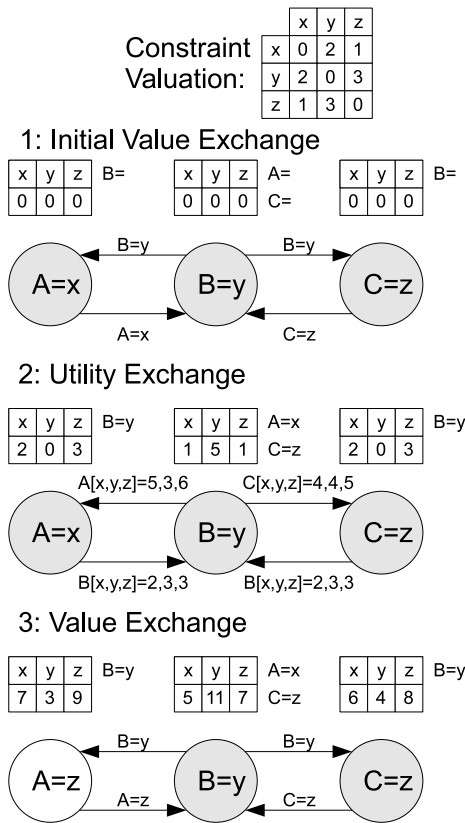


Figure 2: DNEA in action. Given constraint valuations shown at top and a random starting assignment of $A=x$, $B=y$, and $C=z$, DNEA finds the optimal assignment in one round of exchanges. Note how the utility exchange message from B to A in step 2 contains aggregated utility from C.

Utility Exchange

When the agent for variable X receives all new values for neighboring variables of X , it calculates a set of utility exchanges. First, the agent calculates the local utility at X for each possible assignment in D_X to X given the current neighboring variable assignments. Then, for each neighbor Y , the agent calculates its optimal assignment to X for each value in D_Y given the current neighboring variable assignments for all other neighbors than Y . This maximum local utility at X for each value in D_Y is sent to the agent for variable Y . After the agent has calculated and sent all of these utility exchange messages, it waits for updated value exchange messages from its neighbors.

6.2 Analysis

The Distributed Neighborhood Exchange Algorithm is fairly simple to implement and has a similar flow of execution to other local value exchange algorithms, but calculates a merge of utility valuations for all neighbors in a single cycle; this drastically decreases the number of message passing cycles required for convergence. Maximum computation per variable per phase for DNEA is $O(|N_X| \cdot |D_X|)$ for value exchange and $O(\sum_{Y \in N_X} |D_Y| \cdot |D_X|)$ for utility exchange.

7 Application

We have presented a challenging multi-agent coordination problem, a proposal to use DCOP mappings to solve this problem, and an algorithm that will solve the resulting DCOP instance. We now highlight the actual test simulator and some additional support components that we needed to integrate for a full DCOP solution.

7.1 Timed Simulation

The actual Coordinators project uses a simulation framework based on simulated time ticks. A master simulation agent tracks a schedule of agent requests for method executions. The simulation agent sends a pulse message to each agent for each simulated time tick. Actual simulation runs set the simulated time per tick to one second of real time, with the first methods to execute typically available beginning at tick 30. Agent communication is bounded by a message passing infrastructure that allows only around 20-50 messages to be sent from any agent per second. These computational and communication bounds place tight restrictions on the agent reasoning capabilities.

7.2 Support Components

1. *Local Scheduling Agent* - A small piece of code that inspected current DCOP variable assignments and determined when it should request methods to execute. It uses a horizon policy that freezes variable assignments for methods that will be executed in the near future.
2. *Dynamic Distributed Control (DDC)* - A meta-level controller of the DCOP process. Following a similar approach to the one presented in [Zivan, 2008], we create a breadth-first overlay tree on top of the constraint network to propagate global utility. This provides a way to safely randomly restart the underlying DCOP search process to find a better solution over time.

8 Results

We ran tests using the Coordinators simulation for a set of medium sized problems and a set of large sized problems. Both sets were generated in such a way that an offline MDP solver could construct an optimal policy. The medium sized set contains 50 problems with an average of 26 agents, 104 tasks/methods, 16 NLEs, and 112 ticks of execution (problem set "OptOP5PMix"). The large sized set contains 8 problems with an average of 15 agents, 1018 tasks/methods, 137 NLEs, and 1680 ticks of execution (problem set "OptBigReMix"). Results are shown in Figure 3 and Figure 4 as the percentage of optimal quality (as determined by an offline MDP solver) achieved by each approach.

For comparison, we show a baseline strategy that used no coordination and simply executed the initial schedule given in the problem, and opportunistically inserted unscheduled methods when they would not conflict with future scheduled methods (labeled Naive). We also show results for the three teams that took part in the Coordinators second phase testing (labeled using abbreviations from Section 2.1). Our DCOP approach is shown using the Distributed Neighbor Exchange Algorithm (DNEA).

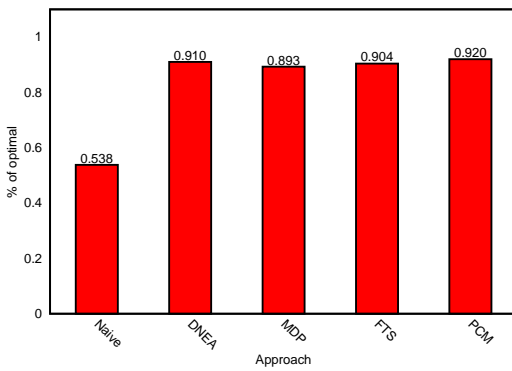


Figure 3: OptOP5PMix: Solution Quality as % of optimal

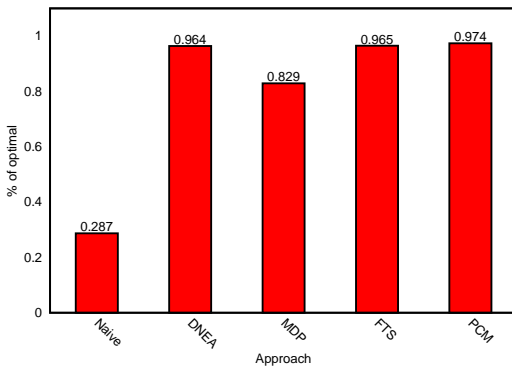


Figure 4: OptBigReMix: Solution Quality as % of optimal

9 Analysis

We see clearly that our DCOP approach can achieve much higher performance than the baseline. Our mapping from CTÆMS to DCOP has succeeded in representing the underlying problem. We also see that DNEA adequately handles large scale DCOP instances with 1000+ variables and meets the tight computation and communication bounds required for the Coordinators simulation. Using a matched pair t-test, and a Wilcoxon signed-rank test for the “OptOP5PMix” set, we find that our solution is statistically the same as the FTS, PCM, and MDP approaches. On the larger “OptBigReMix” set we find that our solution is statistically the same as the FTS and PCM approaches, and is significantly better than the MDP approach ($p < 0.00001$ for matched pair t-test and $p < 0.001$ for Wilcoxon signed-rank test).

10 Conclusion

We have presented a DCOP based solution to a very complex, real-world problem domain. To achieve the fully integrated solution, we developed a new problem representation mapping, introduced extensions to the DCOP formalization for problems with uncertainties, and introduced a new scalable DCOP algorithm that achieved a high level of performance for this domain, comparable to the two best performing Coordinators teams. These results show that DCOP techniques can be used to solve large-scale, real-world problems, and

that continued work in this direction is very promising. Our immediate future work will be to extend our solution to handle dynamic changes to the CTÆMS task structure.

References

- [Atlas and Decker, 2007] James Atlas and Keith Decker. Task scheduling using constraint optimization with uncertainty. In *AAMAS '07 - Workshop on Coordinating Agents' Plans and Schedules*, CAPS, pages 25–28, 2007.
- [Bowring *et al.*, 2008] Emma Bowring *et al.* On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In *AAMAS '08*, pages 607–614, Richland, SC, 2008. IFAAMAS.
- [Farinelli *et al.*, 2008] Alessandro Farinelli *et al.* Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS '08*, 2008.
- [Horling *et al.*, 1999] Bryan Horling *et al.* The TAEMS White Paper, January 1999.
- [Junges and Bazzan, 2008] Robert Junges and Ana L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *AAMAS '08*, pages 599–606, Richland, SC, 2008. IFAAMAS.
- [Maheswaran *et al.*, 2004] Rajiv T. Maheswaran *et al.* Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS '04*, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.
- [Maheswaran *et al.*, 2008] Rajiv T. Maheswaran *et al.* Predictability & criticality metrics for coordination in complex environments. In *AAMAS '08*, pages 647–654, Richland, SC, 2008. IFAAMAS.
- [Modi *et al.*, 2005] P. Modi *et al.* Adopt: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161:149–180, 2005.
- [Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI 05*, pages 266–271, Edinburgh, Scotland, Aug 2005.
- [Smith *et al.*, 2007] Stephen F. Smith, Anthony Gallagher, and Terry Zimmerman. Distributed management of flexible times schedules. In *AAMAS '07*, pages 1–8, New York, NY, USA, 2007. ACM.
- [Sultanik *et al.*, 2007] Evan Sultanik, Pragnesh Jay Modi, and William Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, 2007.
- [Witwicki and Durfee, 2007] Stefan Witwicki and Edmund Durfee. Commitment-driven distributed joint policy search. In *AAMAS '07*, pages 1–8, New York, NY, USA, 2007. ACM.
- [Zivan, 2008] Roie Zivan. Anytime local search for distributed constraint optimization. In *AAMAS '08 - DCR Workshop (Distributed Constraint Reasoning)*, 2008.