Second International Workshop on

Optimisation in Multi-Agent Systems

11th May 2009

co-located with
the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems,
Budapest, Hungary

Editors:

Nicholas R. Jennings
Alex Rogers
Juan Antonio Rodriguez Aguilar
Alessandro Farinelli
Sarvapali D. Ramchurn

Foreword

11^{th} May 2009

The number and variety of applications of multi-agent systems has increased significantly over the last few years, ranging from online auction design, through multi-sensor networks, to scheduling of tasks in multi-actor systems. In many cases, however, the systems designed for these applications require some form of optimisation in order to achieve their goals. Given this, a number of advances have been made in the design of winner determination algorithms, coalition formation techniques, and distributed constraint optimisation algorithms, among others. Nevertheless, there are no general principles guiding the design of such algorithms that would enable researchers to either exploit solutions designed in other areas or to ensure that their algorithms conform to some level of applicability to real problems.

Against this background, and based on the success of the first OPTMAS 08 workshop held at AAMAS in 2008, we present here the contributions to the second workshop on Optimisation in Multi-Agent Systems (OPTMAS 09). Our first aim here is to bring together researchers from different parts of the multi-agent systems research area, to present their work and discuss acceptable solutions, benchmarks, and evaluation methods for generally researched optimisation problems. In this context, this collection ranges across contributions in the are of Distributed Constraint Optimisation, Task Assignment, Game Theory and Learning. The collection aims to support interactions between researchers working on similar optimisation problems or techniques from various areas of the multi-agent systems community.

Nicholas R. Jennings, Alex Rogers, Juan Antonio Rodriguez Aguilar,
Alessandro Farinelli, and Sarvapali D. Ramchurn

11^{th} May 2009.

Table of Contents

Author Index

eXtreme-Ants: Ant Based Algorithm for Task Allocation in Extreme Teams

Fernando dos Santos and Ana L. C. Bazzan
PPGC – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064, CEP 91501-970, Porto Alegre, RS, Brasil
{fsantos,bazzan}@inf.ufrgs.br

ABSTRACT

This paper addresses the problem of multiagent task allocation in extreme teams. An extreme team is composed by a large number of agents with overlapping functionality operating in dynamic environments with possible inter-task constraints. We present an approximate algorithm for task allocation in extreme teams, called eXtreme-Ants. The algorithm is inspired in the division of labor in social insects and in the process of recruitment for cooperative transport observed in ant colonies. The model of division of labor offers fast and efficient decision-making, while the recruitment ensures the allocation of constrained tasks that require simultaneous execution. We show that eXtreme-Ants outperforms other two algorithms regarding communication and computational effort.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms

Keywords

Multiagent Task Allocation, Swarm Intelligence

1. INTRODUCTION

How to efficiently allocate tasks among agents in large-scale and dynamic environments? A large-scale environment means thousands of agents that must coordinate themselves to allocate and perform the available tasks. Scerri *et al.* call these scenarios *extreme teams* [7]. Task allocation in extreme teams is associated with four features: (i) dynamic environments, in which task can appear and disappear; (ii) agents perform multiple tasks given their available resources; (iii) agents have overlapping functionality to perform the tasks but with differing levels of capability; and (iv) inter-task constraints can be present, imposing simultaneous execution requirements.

Extreme teams can be formalized as an extended generalized assignment problem (E-GAP) [7]. The E-GAP model captures precisely the characteristics of extreme teams and defines the solution as the allocation which maximizes a reward measure, given by the capabilities of the agents that take part of the allocation. Efficient multiagent techniques to deal with E-GAP are a prerequisite to build teams of robots to act in extreme situations. Besides the reward, the communication channel must be used in the best way possible to avoid an excessive amount of communication. Moreover, the computational effort employed by the agents to decide which tasks to accept must be as low as possible, enabling they to act in environments where the available time to make a decision is highly restricted.

Social insects (e.g. ants) have the characteristics of extreme teams. Thus, we can conclude that Nature, despite the simplicity of the insects and over years of evolution, has provided these insects with the capability to effectively act in these teams.

To perform the tasks related to the nest survival, social insects adopt a division of labor among workers. Theraulaz *et al.* [8] present a mathematical model to replicate some mechanics of division of labor. This model is based on individual response thresholds and tasks stimuli. Moreover, it is not required that individuals have complete information about the environment and there is no need of team leaders.

Simultaneous execution of tasks also exists in social insects, as for instance in some species of ants. The task in question is the transportation of large preys. Instead of seize and transport individually a large prey, some species form groups of ants to cooperatively transport a prey. These groups are formed via a process called recruitment [2]. In this sense, the large prey can be seen as a set of interdependent subtasks, where each one is simultaneously executed by an ant.

We propose a multiagent approximate task allocation algorithm, called eXtreme-Ants, which is inspired in the division of labor in social insects and in the process of recruitment present in ants. Agents running eXtreme-Ants are efficient to act in extreme teams, with low computational effort and communication. We empirically evaluate eXtreme-Ants in a domain independent simulator and compare it with two other algorithms that are GAP-based: Swarm-GAP [1] and LA-DCOP [7]. The algorithm eXtreme-Ants achieves total rewards close to the ones achieved by LA-DCOP, but with lower communication and computational effort. Regarding Swarm-GAP, eXtreme-Ants yields better total rewards, particularly in the presence of inter-task constraints that impose simultaneous execution.

The remaining of this paper is organized as follows. Section 2 discusses the GAP and its extension (E-GAP), the model of division of labor in social insects, and the process of recruitment used by ants to cooperatively transport large preys. Section 3 details other two algorithms

for dealing with GAP-based task allocation, Swarm-GAP and LA-DCOP. Section 4 presents the proposed algorithm. Section 5 presents the empirical evaluation via a series of experiments. Finally, section 6 points out the conclusion and future directions.

2. BACKGROUND

2.1 GAP and EGAP models

The generalized assignment problem (GAP) is a model used to formalize the multiagent task allocation problem [3]. A GAP is composed by a set $\mathcal{J}$ of tasks to be performed by a set $\mathcal{I}$ of agents. Each agent $i \in \mathcal{I}$ has a capability to perform each task $j \in \mathcal{J}$ denoted by $Cap(i,j) \rightarrow [0,1]$. Each agent also has a limited amount of resource $i.res$ and uses $Res(i,j)$ when performs task j. An allocation matrix M is used to represent the allocation, where m_{ij} is given by Equation 1.

$$m_{ij} = \begin{cases} 1 & \text{if } i \text{ performs } j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The goal is to find M that maximizes the allocation reward, which is given by the agents' capabilities, as shown in Equation 2.

$$M = \underset{M'}{\operatorname{argmax}} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} Cap(i,j) \times m'_{ij} \quad (2)$$

The allocation M must respect all agents' resources limitations (Equation 3) and each task must be allocated to at most one agent (Equation 4).

$$\forall i \in \mathcal{I}, \sum_{j \in \mathcal{J}} Res(i,j) \times m_{ij} \leq i.res \quad (3)$$

$$\forall j \in \mathcal{J}, \sum_{i \in \mathcal{I}} m_{ij} \leq 1 \quad (4)$$

The GAP model was extended by Scerri *et al* [7] to incorporate two features related to extreme teams: scenario dynamics and inter-task constraints. This extended model was called extended generalized assignment problem (E-GAP).

Inter-task constraints are interdependencies among tasks. We focus on AND constraints here, but the formalization can be extended to other constraint types as well. In the case of an AND constraint the agents only receive the reward if all constrained tasks are *simultaneously* executed. The AND constrained tasks can be viewed as a decomposition of a large task into interdependent subtasks. The execution of some subtasks does not leads to the successful execution of the large task, wasting the agents' resources and not producing the desired effect in the system. Moreover, in the case of physical robots, they can be damaged attempting to perform an effort greater than their capabilities (e.g. trying to remove a large piece of collapsed building from a blocked road).

To formalize AND constrained tasks, the E-GAP model defines a set $\bowtie = \{\alpha_1, \ldots, \alpha_p\}$ containing p sets α of AND constrained tasks in the form $\alpha_k = \{j_1 \wedge \ldots \wedge j_q\}$. Each AND constrained task j belongs to at most one set α_k. The number of tasks that are being performed in a set α_k is given by Equation 5.

$$x_k = \sum_{i \in \mathcal{I}} \sum_{j \in \alpha_k} m_{ij} \quad (5)$$

Let $v_{ij} = Cap(i,j) \times m_{ij}$. Given the constraints of $\bowtie$, the reward $Val(i,j,\bowtie)$ of an agent i performing the task j is given by Equation 6.

$$Val(i,j,\bowtie) = \begin{cases} v_{ij} & \text{if } \forall \alpha_k \in \bowtie, j \notin \alpha_k \\ v_{ij} & \text{if } \exists \alpha_k \in \bowtie \text{ with } j \in \alpha_k \wedge x_k = |\alpha_k| \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

To represent the dynamics in the scenario all E-GAP variables are indexed by a time step t. The goal is to find a sequence of allocations $\overrightarrow{M}$ one for each time step t, as shown in Equation 7. A delay cost function $DC^t(j^t)$ can be used to define the cost of not performing a task j at time step t.

$$\begin{aligned} f(\overrightarrow{M}) = & \sum_t \sum_{i^t \in \mathcal{I}^t} \sum_{j^t \in \mathcal{J}^t} (Val^t(i^t, j^t, \bowtie^t) \times m^t_{ij}) \\ & - \sum_t \sum_{j^t \in \mathcal{J}^t} (1 - \sum_{i^t \in \mathcal{I}^t} m^t_{ij}) \times DC^t(j^t) \end{aligned} \quad (7)$$

Furthermore, the agents' resource limitations must be respected at each time step t (Equation 8) and each task must be allocated to at most one agent (Equation 9).

$$\forall t, \forall i^t \in \mathcal{I}^t, \sum_{j^t \in \mathcal{J}^t} Res^t(i^t, j^t) \times m^t_{ij} \leq i^t.res \quad (8)$$

$$\forall t, \forall j^t \in \mathcal{J}^t, \sum_{i^t \in \mathcal{I}^t} m^t_{ij} \leq 1 \quad (9)$$

2.2 Division of Labor in Social Insects

An effective division of labor is responsible for the ecological success of insect societies. A social insect colony with hundreds of thousand members operates without the existence of explicit coordination. An individual cannot assess the needs of the colony; it just has a fairly simple local information, and no one is in charge of coordination. From individual workers aggregation, the colony behavior emerges without any type of explicit coordination or planning. The key point is the plasticity of the individuals, in other words, the existence of a behavioral flexibility. This flexibility allows the individuals to engage in different tasks responding to changing conditions in the colony.

Observations regarding this behavior are the basis of the theoretical model described by Theraulaz *et al.* [8]. In this model, interactions among members of the colony and individual perception of local needs result in a dynamic distribution of tasks. The model is based on individuals' internal response threshold related to tasks stimuli. Assuming the existence of $\mathcal{J}$ tasks to be performed, each task $j \in \mathcal{J}$ has an associated stimulus s_j. The stimulus is related to the demand for the task execution, and can be a number of encounters, a chemical concentration, or any quantitative cue sensed by individuals. Given a set of $\mathcal{I}$ individuals which can perform the tasks of $\mathcal{J}$, each individual $i \in \mathcal{I}$ has an internal response threshold θ_{ij}, which is related to the likelihood of reacting to the stimulus associated with task j.

The threshold can be seen as a genetic characteristic (also called polymorphism, which is responsible for the existence of differences in the morphologies of insects belonging the same society), or as a temporal polyethism (in which individuals of the same age tend to perform identical sets of tasks), or simply as individual variability.

In the model of Theraulaz *et al.* [8] the individual internal threshold θ_{ij} and the task stimulus s_j represent the proba-

bility (tendency) $T_{ij}(s_j)$ of the individual i to perform task j, as shown in Equation 10.

$$T_{ij}(s_j) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} \quad (10)$$

This tendency means that any individual is able to perform any task if the corresponding task stimulus is high enough to overcome the individual's internal threshold. This flexibility enables the survival of the colony in an eventual absence of specialized individuals, since other individuals start to perform the tasks when the stimulus exceeds their thresholds.

2.3 Recruitment for Cooperative Transport

In some species of ants the transportation of large preys in a cooperative way involves two or more ants that cannot do the transport alone [6]. The main purpose of the cooperative transport is to maximize the trade off between the gained energy (food) and the energy spent to take it to the nest. Further, this process speeds up the transport.

The group involved in the cooperative transport is formed by a process called *recruitment*. When a single scout ant discovers a prey, it firstly attempts to seize and transport it individually. After unsuccessful attempts, a recruitment process starts. To recruit nestmates the ants employ a mechanism called long-range recruitment (LRR). Some species also employ a second mechanism, called short-range recruitment (SRR). In both mechanisms the ants use communication through the environment (stigmergy) [2].

In the SRR the scout that discovers the prey releases a secretion. Shortly thereafter, nestmates in the vicinity are attracted to the prey location by the secretion odor. When the prey cannot be moved by the scout and the ants recruited via SRR, one of the ants begins the LRR. Hölldobler *et al* report that SRR is sufficient to summon enough ants to transport the prey in the majority of the cases [2].

In the LRR the scout ant that discovers the prey returns to the nest to recruit nestmates. In the course towards the nest, the scout lays a pheromone trail. Nestmates encountered in the course are stimulated by the scout via direct contact. After, the stimulated nestmate also begins to lay a pheromone trail even though it had not yet experienced the prey stimulus itself, thus establishing a chain of communication among nestmates. When the scout arrives at the nest, nestmates are attracted by the pheromone and run to the prey site.

After the recruited ants arrive where the prey is, they begin the cooperative transport. The number of ants engaged in the transport is regulated at the prey site and depends on its characteristics, such as weight, size, rotational forces, and difficulty to move. When the number of ants present is not enough to move the prey, more ants are recruited by one of the aforementioned processes, until the prey is successful transported. Although in a more economic approach the scout ant should recruit an exact number or nestmates, it was suggested that the scout cannot make a fine assessment of the number of ants required to retrieve the prey [6]. Therefore, the most effective strategy may be to recruit a constant number of ants followed by a regulation of the group size during the transport.

In summary, the recruitment for cooperative transport consists in three steps:

1. The scout ant that discovered the prey starts the recruitment, inviting netstmates with pheromones;

2. The ants that accept to join the recruitment move to the prey site;

3. The size of the transportation group is regulated to the prey characteristics.

3. RELATED WORK

The research regarding multiagent task allocation has shown significant advances in the last few years (auction, contracting, coalition formation, organizations, etc). A complete review of the subject is outside the scope of this paper. We just mention that auctions are normally centralized mechanisms, in which agents put bids to an auctioneer, depending on their capabilities and resources. After receiving all bids, the auctioneer makes the allocation of the tasks among the bidders. Centralized auctioneers can have severe bottlenecks. Further, auctions require high amounts of communication [9].

Here we concentrate in the line of research that deals with coordination for task allocation. Within this research line, one approach is the framework of distributed constraint optimization problem (DCOP). A DCOP consists of a set of variables that can assume values from a discrete domain. Each variable is assigned to one agent which has the control over its value. The goal of the agents is to choose values for the variables to optimize a global objective function. This function can be described as an aggregation over a set of cost functions related to pairs of variables. A DCOP can be represented by a constraint graph, where vertices are variables and edges are cost functions between variables. Despite the existence of complete algorithms for DCOP, such as Adopt [4], and DPOP [5], these are not efficient to deal with the problem of multiagent task allocation. Due to dense constraint graphs generated to represent the problem, Adopt and DPOP demand high communication and space respectively.

To deal with the particular characteristics of extreme teams, Scerri *et al.* present an approximate algorithm called Low-communication Approximate DCOP (LA-DCOP), which uses tokens to represent tasks and further minimize communication [7]. An agent decides whether or not to accept a task based both on its capability and on a threshold associated to the task. To deal with inter-task constraints, LA-DCOP uses a differentiated kind of token, called potential token. If an agent in LA-DCOP is able to allocate more than one task, it must select the ones that maximize its capability given its resources. This selection is a maximization problem, which can be reduced to a binary knapsack problem (BKP), proved to be NP-Complete. The computational complexity of LA-DCOP thus depends of the complexity of its function to deal with the BKP.

Another approximate algorithm which can deal with extreme teams is the Swarm-GAP [1]. An agent in Swarm-GAP decides whether or not to accept a task based on the model of division of labor used by social insects colonies. This algorithm also uses tokens to represent tasks. To deal with inter-task constraints, the agents in Swarm-GAP just increase the tendency to allocate a constrained task by a factor called execution coefficient. The execution coefficient is computed using the rate between the

number of constrained tasks which are allocated and the total number of constrained tasks.

4. eXtreme-Ants

4.1 Basic Ideas

eXtreme-Ants is an approximate algorithm that solves E-GAPs. Agents running eXtreme-Ants use the model of division of labor in social insects (Equation 10) to decide whether or not to perform the tasks. The notation used hereafter to represent agents, tasks, and all other terms is the one from the E-GAP model (Section 2.1) and from the model of division of labor (Section 2.2). The internal threshold θ_{ij} of an agent i related to a task j is defined via the concept of polymorphism and corresponds to the inverse of the capability $Cap(i, j)$, as shown in Equation 11. If an agent is not capable regarding a particular task, then its internal threshold is set to infinity, avoiding the allocation of the task to the agent. This makes sense if we consider the capability as a kind of morphism. For example, a fire brigade agent is more capable of fighting fires than rescuing civilians. Thus it have low thresholds related to fire fighting tasks and high thresholds to rescue civilians.

$$\theta_{ij} = \begin{cases} 1 - Cap(i,j) & \text{if } Cap(i,j) > 0 \\ \infty & \text{otherwise} \end{cases} \quad (11)$$

Each task $j \in \mathcal{J}$ has an associated stimulus s_j. The stimulus controls the allocation of the tasks by the agents. Low stimuli mean that the tasks will only be accepted by agents with low thresholds (thus, more capable). High stimuli increase the chance of the tasks to be accepted, even by agents with high thresholds (less capable).

Since in the E-GAP each task must be allocated to at most one agent, eXtreme-Ants uses tokens to represent the tasks and ensure this mutual exclusion constraint. A token contains a list of tasks it represents. An agent that holds a token has the exclusive right to accept the tasks contained in a token. If the agent does not accept all tasks, it passes the token to another teammate. In this way, eXtreme-Ants avoid conflicts in the allocation and reduces the communication.

To deal with AND constraints among tasks, agents in eXtreme-Ants reproduce the recruitment process of ants. When an agent detects that it is not capable of accepting all AND constrained tasks perceived, it recruits other agents to form a group committed with the simultaneous execution.

4.2 Algorithm Details

Algorithms 1 and 2 present the details of our approach. Each agent i reacts to two events: perception of a set of tasks (which can be AND constrained), and receipt of messages. In the following we detail the algorithm operation.

When the agent perceives a set $\mathcal{J}$ of tasks (line 1) it creates a token to store the perceived tasks. The agent then decides whether or not to accept the tasks contained in the token, given its tendency and the available resources (lines 21-30). When a task is allocated to agent i, the available resources at i is decreased by the amount required. If some tasks contained in the token remains unallocated, the agent sends the token to a randomly selected teammate. As in [7], to avoid agents passing token back and forth, each token maintains a list of visited agents. The token can revisit an agent only after all were visited.

Algorithm 1: eXtreme-Ants for agent i

```
1  when perceived set of tasks J
2      token := newToken();
3      add each j ∈ J to token.tasks;
4      evaluateToken(token);
5  end

6  when perceived set of AND constrained tasks α_k
7      /* firstly try to accept all tasks by itself */
8      foreach j ∈ α_k do
9          if roulette() < T_ij and i.res ≥ Res(i, j) then
10             accept task j and decrease i.res;
11         end
12     end
13     if there are non accepted tasks in α_k then
14         discard previous accepted tasks of α_k (lines 7-9);
15         performsRecruitment(α_k);
16     end
17 end

18 when received token
19     evaluateToken(token);
20 end

21 procedure evaluateToken(token)
22     /* decides whether or not to accept the tasks */
23     foreach j ∈ token.tasks do
24         if roulette() < T_ij and i.res ≥ Res(i, j) then
25             accept task j and decrease i.res;
26             token.tasks := token.tasks − j;
27         end
28     end
29     if there are non accepted tasks in token.tasks then
30         send token to a teammate;
31     end
32 end
```

When the agent perceives a set of AND constrained tasks α_k (line 6), it acts as a scout ant. Firstly it attempts to accept all the constrained tasks. If it fails, it begins a recruitment process. We develop a protocol that reproduces the three steps of the recruitment process of ants via the use of messages. There are five kinds of messages used in the recruitment protocol of eXtreme-Ants:

request: to invite an agent to join the recruitment for a task $j \in \alpha_k$ and to commit to it;

committed: to inform that the agent joins the recruitment for a task $j \in \alpha_k$ and commits to it;

engage: to inform that the agent was indeed selected to perform the task $j \in \alpha_k$;

release: to inform that the agent was not selected to perform the task $j \in \alpha_k$ and must uncommit with it.

timeout: to inform that a request for a task $j \in \alpha_k$ reaches its timeout.

For the first step of the recruitment, the scout agent sends a certain number of **request** messages for each task $j \in \alpha_k$ (lines 33-37). These requests are sent to randomly

Algorithm 2: eXtreme-Ants for agent i (cont.)

```
33 procedure performsRecruitment(AND set α_k)
34     repeat
35         j := pick a task from α_k;
36         send("request", j) to a teammate;
37     until the maximum number of requests sent for all
       j ∈ α_k is reached or the recruitment for α_k is
       finished or aborted.
38 end

39 when received ("request", j) from agent i_s
40     /* decides whether or not to commit */
41     if roulette() < T_ij and i.res ≥ Res(i, j) then
42         commit to j;
43         send("committed", j) to i_s;
44     else
45         if request timeout reached then
46             send("timeout", j) to i_s;
47         else
48             forward("request", j) to a teammate;
49         end
50     end
51 end

52 when received ("committed", j) from i_c
53     α_k := AND group which contains j;
54     if recruitment for α_k is finished or aborted then
55         send("release", j) to i_c; return
56     end
57     if at least one agent committed with each j ∈ α_k
       then
58         recruitment for α_k is finished;
59         /* forms the group of engaged agents */
60         foreach j ∈ α_k do
61             pick a committed agent i_p with probability
               proportional to Cap(i_p, j);
62             send("engage", j) to i_p;
63             send("release", j) to non selected agents;
64         end
65     end
66 end

67 when received ("engage", j)
68     accept task j and decrease i.res;
69 end

70 when received ("release", j)
71     uncommit to j;
72 end

73 when received ("timeout", j)
74     α_k := AND group which contains j;
75     if the number of received timeouts for each j ∈ α_k is
       equal the number of requests sent then
76         recruitment for α_k is aborted;
77         foreach j ∈ α_k do
78             send("release", j) to committed agents;
79         end
80     end
81 end
```

selected teammates and act as the pheromone released in the air (SRR) or released on the way to the nest (LRR). As it occurs with the scout ant, which recruits a fixed number of nestmates independently of prey characteristics, eXtreme-Ants fixes a maximum number of requests that must be sent for each AND constrained task. This maximum number must be experimentally determined to maximize the total reward.

In the second step, the agents must decide whether or not to join the recruitment. When an agent receives a request originated by a scout agent i_s for a task j (lines 39-48), it uses the tendency (Equation 10) to decide if it accepts the request and then joins the recruitment, avoiding double commitment. If the request is accepted, the agent commits to perform the task, reserving the amount of resources required by the task. A **committed** message is send to the scout to inform the commitment. If the request is not accepted, the agent forwards it to another randomly selected teammate, reproducing the chain of communication present in the LRR.

In the third step, the size of the group of agents engaged in the simultaneous execution of the AND constrained tasks must be regulated. In eXtreme-Ants the regulation is done by the scout agent. When the scout receives enough commitments for each constrained task $j \in \alpha_k$ (line 57), it forms the group of agents which will execute the tasks simultaneously. Following the E-GAP definition, just one agent must be selected among those committed for each constrained task. The scout then performs a probabilistic selection, picking an agent i_p with probability proportional to its capability $Cap(i_p, j)$. The scout then informs i_p that it was the selected one and thus must engage in the execution of j (via an **engage** message, line 62). All other non selected agents are released (via a **release** message, line 63). Agents that commit to an already allocated task are also released to avoid deadlocks. At this moment the recruitment is finished. As the result, a group of agents is formed, in which each agent is allocated to a task $j \in \alpha_k$, enabling the simultaneous execution of all AND constrained tasks in α_k.

After the group of engaged agents is formed, the requests not yet accepted by some agent become obsolete. To avoid agents passing obsolete requests back and forth, eXtreme-Ants introduces a timeout mechanism. The timeout is a number of agents that a recruitment request is allowed to visit. When the timeout of a request is detected (line 45), the scout is notified via a **timeout** message. When the scout agent receives a timeout notification for all requests sent, it aborts the recruitment and releases the committed agents.

It is important to note that due to the algorithm asynchronism, the scout agent can perform another actions while the recruitment occurs. These actions comprise the perception of another tasks, and even a recruitment for other AND constrained task groups. Although eXtreme-Ants reproduce the inter-agent communication via messages, it can be easily modified to use some kind of indirect communication (e.g. pheromones) when the environment allows it.

5. EXPERIMENTS AND RESULTS

We compare eXtreme-Ants to Swarm-GAP[1] and LA-DCOP[7]. We have evaluated eXtreme-Ants in a domain independent simulator that allows experimentation with large number of agents and tasks, performing exper-

iments similar to Swarm-GAP and LA-DCOP which have also used such a simulator.

Basically, each experiment consists of 2000 tasks, grouped in five classes, where each class determines the task characteristics. The number of agents varies from 500 to 4000. This means that the load (ratio between tasks and agents) is 4 in the first case and 0.5 in the latter. Each agent has a 60% probability of having a non-zero capability for each class. In this case the agent has a randomly assigned capability ranging from 0 to 1. Regarding the AND constraints among tasks, 60% of the tasks are related in groups of five tasks. The simulated communication channel is reliable (every sent message is received) and fully connected (each agent is connected to every other agent). Each experiment consists of 1000 time steps. The total number of tasks is kept constant. At each time step, each task has a probability of 10% to be replaced by a task potentially requiring a different capability. The tasks are persistent, which means that non allocated tasks are kept in the next time step. At each time step, each token or message is allowed to move from one agent to another only once. Despite that each task can have a particular stimulus value, we adopt the same value for all tasks. Each datapoint in the plots we show here represents the average over 20 runs. The standard deviations are not shown due to their low values.

As defined by the E-GAP, the goal is to maximize the total reward, which is the sum of the reward at each time step over the length of the simulation. The first experiment compares the total reward achieved by each algorithm. The parameters used for each algorithm are shown in Table 1. These parameters, which were selected among a large set of tested values, yield the maximum total reward in each scenario, and will be used in the comparisons. Additionally, in the case of eXtreme-Ants the total rewards are obtained with five recruitment requests for each AND constrained task, and with a timeout of 20 visited agents.

Table 1: Parameter values that yield the maximum total reward for each algorithm.

Agents	eXtreme-Ants (Stimulus)	Swarm-GAP (Stimulus)	LA-DCOP (Threshold)
500	0.3	0.2	0.0
1000	0.3	0.3	0.4
1500	0.2	0.2	0.6
2000	0.2	0.2	0.6
2500	0.2	0.2	0.6
3000	0.2	0.2	0.6
3500	0.2	0.2	0.7
4000	0.2	0.2	0.7

Figure 1 shows the total rewards achieved by each algorithm. On average, eXtreme-Ants yields rewards that are 25% higher than those of Swarm-GAP and 19% lower than those of LA-DCOP (t-test, 99% confidence).

When an agent accepts a task, it uses an amount of its resources. Thus, the agents must avoid to waste their resources accepting tasks that do not yield any reward (e.g. tasks that belong to an AND constrained, but are not simultaneously accepted). The second experiment, shown in Figure 2, compares the percentage of resources used by each agent to accept tasks at each time step. As we can see, all algorithms use almost the same percentage of resources. There is no significative difference between

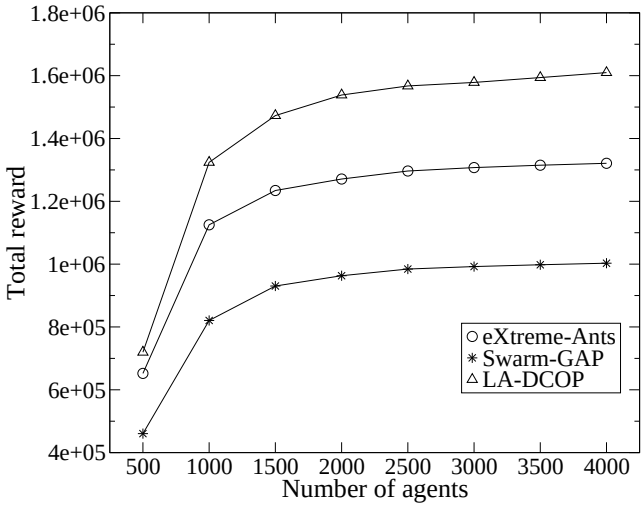


Figure 1: Total reward versus the number of agents.

eXtreme-Ants and Swarm-GAP in the cases with 3000, 3500, and 4000 agents, and between eXtreme-Ants and LA-DCOP in the cases with 500 and 1000 agents (t-test, 99% confidence).

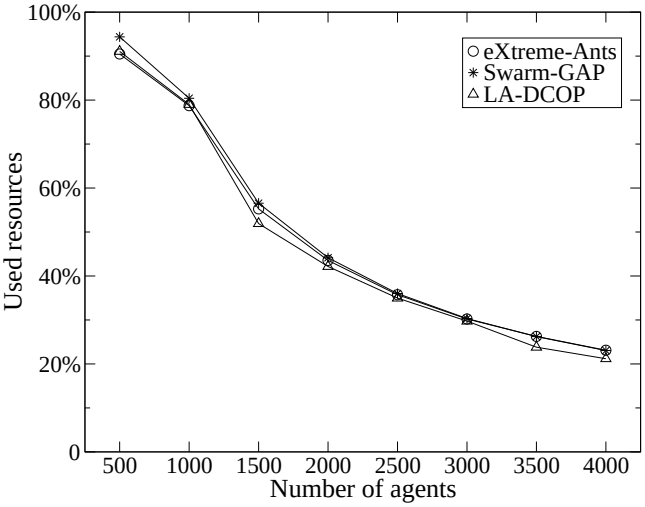


Figure 2: Percentage of resources used by each agent at each time step to allocate tasks.

From these two experiments, we can see that despite the fact that agents in Swarm-GAP use almost the same percentage of resources, the achieved total rewards are worse than those achieved by eXtreme-Ants and LA-DCOP. This is due to the way Swarm-GAP deals with AND constrained tasks. The use of an execution coefficient (see Section 3) does not ensure the simultaneous allocation of the AND constrained tasks. Thus, the agents use their resources to accept tasks, but this allocation does not translate into a reward. Both eXtreme-Ants and LA-DCOP outperform Swarm-GAP regarding the total reward. This is due to the existence of explicit coordination mechanisms to deal with AND constrained tasks, ensuring their simultaneous allocation.

LA-DCOP yields higher rewards than eXtreme-Ants be-

cause each agent maximizes its capability when accepting the tasks, taking into account the available resources. On the other hand, agents in eXtreme-Ants make a simple one-shot decision to allocate tasks. The maximization leads to a better exploitation of the agents' resources. However, as we show in the next experiments, there is a tradeoff between the achieved reward and the communication/computational effort.

In the next experiment, shown in Figure 3, we compare the amount of communication used in each algorithm. The communication is measured as the sum of messages sent by the agent over all time steps, regardless of message type (e.g. token, recruitment request, etc.). The results are statistically significant at 99% confidence t-test. On average, agents in eXtreme-Ants sent 121% fewer messages than those in LA-DCOP and 80% more messages than those in Swarm-GAP. The smallest difference to LA-DCOP occurs with 3000 agents. Even in this case LA-DCOP sends 66% more messages than eXtreme-Ants.

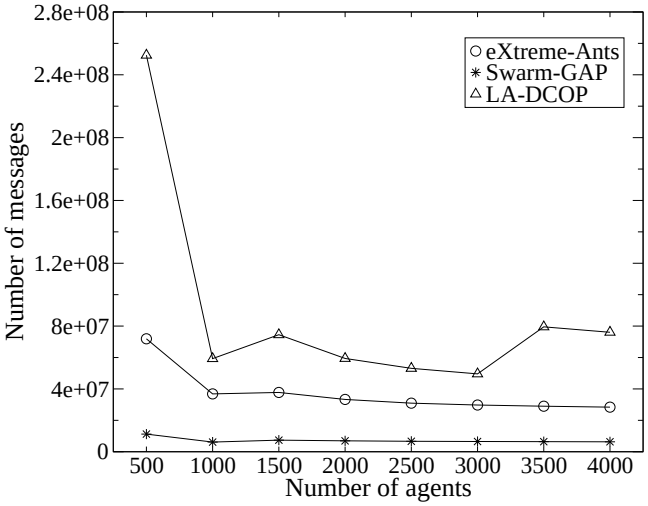


Figure 3: Total number of messages sent versus the number of agents.

As mentioned Swarm-GAP sends fewer messages than eXtreme-Ants and LA-DCOP due to its difficulty to deal with AND constrained tasks. The absence of an explicit coordination mechanism to ensure the simultaneous allocation leads to a small number of messages, but has a great impact in the total reward of Swarm-GAP.

The last experiment aims at evaluating the computational effort of the agents in each algorithm. We define the computational effort as the number of evaluated tasks by each agent at each time step. This number is computed as follows. Each time an agent decides whether or not to accept a task, an internal counter is incremented. In the case of eXtreme-Ants and Swarm-GAP, each probabilistic decision causes just one increment in the counter. On the other hand, since an agent in LA-DCOP solves a BKP to decide which tasks to accept, the increment in the counter is related to the number of retained tasks. To solve a BKP our implemented version of LA-DCOP uses a greedy approach, which sorts the tasks by the agent's capability and then selects the tasks to accept constrained by the agent's resources. If n is the number of retained tasks, the sort causes a increment of $n\ log\ n$ in the counter, followed by a increment of at most n to select the accepted tasks.

Low computational effort means that the agents are more efficient to act in environments in which the available time to make a decision is restricted. Figure 4 shows the average computational effort of each agent at each time step. The external plot emphasizes the area which concentrates the majority of the points. The internal plot shows the full area just to present the points not shown in the external plot. The results are statistically significant at 99% confidence t-test.

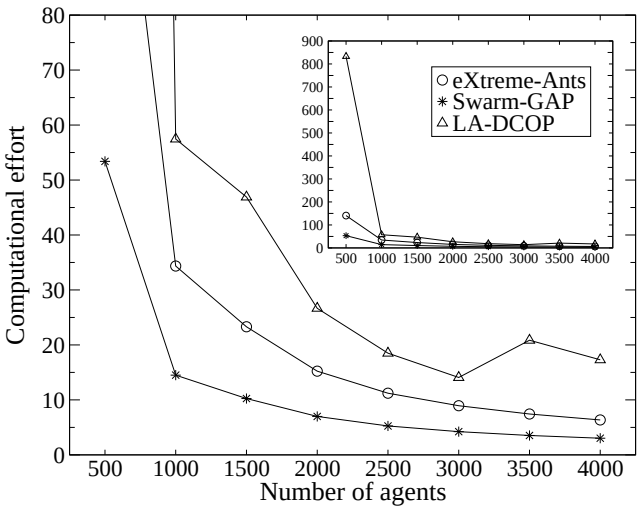


Figure 4: Computational effort as the number of evaluated tasks by each agent at each time step.

The computational effort of Swarm-GAP is, on average, 55% lower than those from eXtreme-Ants. Since in Swarm-GAP there is no explicit coordination mechanism to deal with AND constrained tasks, the agents do not have to make additional evaluations regarding the simultaneous allocation of constrained tasks, reducing the computational effort of Swarm-GAP. However, the absence of such mechanism affects the total reward, as shown previously. The higher computational effort of both eXtreme-Ants and LA-DCOP are due to the presence of an explicit coordination mechanism to deal with constrained tasks.

eXtreme-Ants outperforms LA-DCOP, with computational efforts on average 151% lower than those from LA-DCOP. The most significant result is for the case of 500 agents, in which the computational effort of LA-DCOP is 493% higher than that of eXtreme-Ants.

As shown in the experiments, the probabilistic allocation of eXtreme-Ants, based on the model of division of labor, reduces the amount of communication and the computational effort. The reduction in the computational effort is due to the simple one-shot decision, which does not require any local maximization. The low computational effort causes the reduction in the number of messages sent, since in LA-DCOP the tasks which are not selected in the local maximization are sent do other agents. In both eXtreme-Ants and LA-DCOP the presence of an efficient coordination mechanism to deal with inter-task constraints leads to better total rewards regarding Swarm-GAP.

Finally, we emphasize that the choice of one particular

algorithm must be related with the constraints of the scenario. When the total reward is a key point and there are no constraints in the communication and in the time the agents have to make a decision, LA-DCOP is a good choice. On the other hand, in scenarios with such constraints eXtreme-Ants is more appropriate. It achieves low total rewards, but the decision is faster and there is a better use of the communication channel.

6. CONCLUSIONS

In this paper we have presented a multiagent approximate algorithm for task allocation in extreme teams, called eXtreme-Ants. The algorithm is inspired in the division of labor in social insects and in the process of recruitment present in ants that transport preys cooperatively.

The experimental results show that the use of the model of division of labor to decide whether or not to allocate the tasks allows the agents to make reasonable coordinated actions. Since the decision is probabilistic, it is fast, efficient, and requires a reduced communication and computational effort, enabling the agents to act in environments where the available time to make a decision is highly restricted. Moreover, the incorporated recruitment process provides efficient allocation of constrained tasks that requires simultaneous execution. This avoid that agents waste they resources and leads to better total rewards. The efficiency of eXtreme-Ants regarding communication and computational effort suggests that techniques which are inspired in social insects can be considered for multiagent task allocation.

We intend to work in the direction of changing the stimuli values dynamically, indicating different priorities in the execution of the tasks. More than one kind of resource for an agent can also be considered. Besides, these resources can change over time, as for instance, a battery charge of a robot. We also intend to evaluate the performance in unreliable communication channel, with failures and noises, and to apply this approach in the RoboCup Rescue simulator.

7. ACKNOWLEDGMENTS

This research is partially supported by the Air Force Office of Scientific Research (AFORS) (grant number FA9550-06-1-0517) and by the Brazilian National Council for Scientific and Technological Development (CNPq).

8. REFERENCES

[1] P. R. Ferreira, Jr., F. Boffo, and A. L. C. Bazzan. Using swarm-gap for distributed task allocation in complex scenarios. In N. Jamali, P. Scerri, and T. Sugawara, editors, *Massively Multiagent Systems*, number 5043 in Lecture Notes in Artificial Intelligence, pages 107–121. Springer, Berlin, 2008.

[2] B. Hölldobler, R. C. Stanton, and H. Markl. Recruitment and food-retrieving behavior in *Novomessor* (formicidae, hymenoptera). *Behavioral Ecology and Sociobiology*, 4(2):163–181, 1978.

[3] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, NY, USA, 1990.

[4] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 161–168, New York, USA, 2003. ACM Press.

[5] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland, August 2005. Professional Book Center.

[6] S. K. Robson and J. F. A. Traniello. Resource assessment, recruitment behavior, and organization of cooperative prey retrieval in the ant *Formica schaufussi* (hymenoptera: Formicidae). *Journal of Insect Behavior*, 11(1):1–22, 1998.

[7] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 727–734, New York, USA, 2005. ACM Press.

[8] G. Theraulaz, E. Bonabeau, and J. Deneubourg. Response threshold reinforcement and division of labour in insect societies. In *Royal Society of London Series B - Biological Sciences*, volume 265, pages 327–332, 2 1998.

[9] Y. Xu, P. Scerri, K. Sycara, and M. Lewis. Comparing market and token-based coordination. In *Proc. of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1113–1115, New York, NY, USA, 2006. ACM.

Multiagent Policy Teaching

Lachlan Dufton
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
ltdufton@cs.uwaterloo.ca

Kate Larson
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
klarson@cs.uwaterloo.ca

ABSTRACT

Recently Zhang and Parkes [11, 12] introduced the idea of value-based policy teaching. In their framework, an interested party is able to provide incentives, by changing the environment, in order to encourage an agent to follow a particular policy. In this paper, we extend the Zhang-Parkes framework to a multiagent setting where all agents are in a common environment so that any modifications made by the interested party are experienced by all agents. We characterise when it is possible for the interested party to provide incentives so that all agents follow a particular desired policy. For the case where the interested party is unable to induce all agents to follow a particular desired policy, we propose that a *behaviour-based* policy comparison approach be used, where the interested party maximises the *similarity* of each agent's behaviour to some target behaviour.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Design, Performance

Keywords

Multiagent Learning, Co-ordination, Decision/Utility Theory, Markov Decision Processes

1. INTRODUCTION

Recently Zhang and Parkes [11, 12] introduced the idea of value-based policy teaching. In their framework, an interested party is able to provide incentives, by changing the environment, in order to encourage an agent to follow a particular policy. The focus of these papers was on a single-agent setting, and the authors studied computationally tractable methods for finding incentives.

In this paper, we study the problem of extending the environment design framework to a multiagent setting. In particular, we focus on the case where agents do not interact, but act in a common environment. Website management is a real-world example of such a scenario. The developer of a website may want visitors to view certain key pages in a particular order. Through well-chosen modifications to the website, it may be possible to lead many, and possibly all, visitors to act as the developer desires. Each visitor has different opinions and goals and none interact with each other, but the website is the same for all. The main challenge we examine in this paper is discovering the types of modifications or incentives that lead a heterogeneous collection of agents to act as desired.

In this paper, the agents are modelled as simple planners in a Markov Decision Process (MDP), and an interested party has a desired behaviour for these agents. While the interested party is unable to directly modify the behaviour of each agent, it can modify the environment, and thus agent behaviour.

We provide a linear programming solution for finding a minimal environmental modification that teaches a specific MDP policy to all agents. This is not always possible, and we show how to detect the feasibility of teaching a policy as well as a method for teaching the largest feasible subset of agents. In this case, attempting to teach the policy exactly can be inflexible. An interested party may not be interested in teaching an exact policy, but simply a policy with similar behaviour to a target policy. In this paper we look at how to quantify this behavioural similarity, based on short-term through to long-term behaviour.

This paper begins with an overview of policy teaching and environment design in the single agent setting, and a summary of other related work. Next, we discuss our multiagent model and our solution to the problem of teaching a specific policy to a set of agents. We then examine a value-based approach to policy teaching, that changes the policies of agents to maximise the aggregate value of each agents' policy. We present two methods of individual policy appraisal, based on the states visited when following the policy or by behavioural similarity to a target policy. We also provide a method of finding incentives that maximise the similarity of all agents' behaviours to the target behaviour. This is followed by empirical tests of our new methods and finally conclusions and directions for future work.

2. BACKGROUND

In this section we discuss the problem of policy teaching through the method of environment design. This begins with an overview of the problem in a single-agent setting. The single-agent model forms the foundation of our multiagent model and our work in this paper extends the solu-

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

tion for this single-agent problem. We also examine some other works related to the problem of achieving a desired behaviour in a setting with multiple agents.

2.1 Single Agent Environment Design

There are existing methods that modify an agent's environment to teach a specific policy to the agent [11] or to teach a policy that maximises the value of the policy from the perspective of an interested party [12]. In the single agent setting, the environment is simply a finite-horizon Markov Decision Process (MDP), M, and the agent follows the optimal policy, π. Formally, $M = \{S, A, R, P, \gamma\}$, where

- S is the finite set of states.
- A is the finite set of actions.
- $R : S \rightarrow \mathbb{R}$ is a reward function, giving the agent's utility for each state. This can be represented as a vector $R \in \mathbb{R}^{|S|}$.
- $P : S \times A \times S \rightarrow [0, 1]$ is the state probability transfer function. If the agent plays action a in state s_i, then the probability of transitioning to state s_j is $P(s_i, a, s_j)$.
- $\gamma \in (0, 1)$ is the discount factor.
- $\pi : S \rightarrow A$ is the policy that determines the action taken by the agent at each state.

In direct policy teaching, the interested party (IP) knows the MDP, M, and has a single target policy for the agent, π_t. The goal of the IP is to find some set of state incentives, $\Delta \in \mathbb{R}^{|S|}$, such that the optimal policy of the MDP $M_{R+\Delta} = \{S, A, (R+\Delta), P, \gamma\}$ is π_t. These incentives model environment modifications that increase the agent's reward for each state by the incentive amount.

Value based policy teaching [12] does not have a specific target policy for the agent, but instead uses a reward function for the IP, $G \in \mathbb{R}^{|S|}$. While the agent forms a policy that maximises the expected sum of discount rewards on $M_{R+\Delta}$, the interested party values the policy according to the expected sum of discounted rewards on $M_G = \{S, A, G, P, \gamma\}$.

In both direct and value based policy teaching, incentives are limited to *admissible incentives* [11]. This limits individual incentives to be non-negative (i.e. no punishments) and the expected discounted sum of incentives must be no greater than some value D_{max}. Stated formally, Δ is admissible if it satisfies the following constraints:

$$\Delta(s) \geq 0 \qquad \forall s \in S$$
$$V_\Delta^\pi(start) \leq D_{max}$$

where the expected discounted sum of incentives $V_\Delta^\pi(s)$ for an agent's policy π is defined as:

$$V_\Delta^\pi(s) = \Delta(s) + \gamma \sum_{s' \in S} P(s, \pi(s), s') V_\Delta^\pi(s') \quad \forall s \in S \quad (2)$$

Zhang and Parkes provided a linear programming solution for direct policy teaching with environment design [11]. This solution minimises the expected incentive spending, $V_\Delta^{\pi_t}(start)$, while satisfying constraints both for admissibility and those found through inverse reinforcement learning (IRL) [7]. IRL takes a policy, π, and an MDP without reward function, M_{-R} and calculates a space of reward functions, $IRL^\pi \subseteq \mathbb{R}^{|S|}$, such that any reward function in this space has an optimal policy π. This space is defined by the following constraints:

$$(P_{\pi_t} - P_a)(I - \gamma P_{\pi_t})^{-1} R \succeq 0 \ , \forall a \in A$$

where P_a and P_{π_t} are the state probability transfer matrices for action a and target policy π_t, respectively. The target policy becomes *uniquely* optimal in $IRL^\pi \subseteq \mathbb{R}^{|S|}$ if the inequality is replaced with a strict inequality.

The direct policy teaching linear program finds the incentive function with minimal expected cost that, when added to the agent's rewards, is in the space of reward functions determined by IRL. Zhang and Parkes also provide a mixed integer program for single-agent, value-based policy teaching [12]. Instead of minimising incentive costs, this program maximises the interested party's value of the policy taken by the agent. However, for the remainder of this paper we will focus on *multiagent* policy teaching.

2.2 Related Work

At a high level, multiagent policy teaching is related to the game theoretic technique of *mechanism design* [8]. Mechanism design seeks to achieve certain behaviours in the agents of a system by defining the rules of the system. This is similar to policy teaching and environment design, which seeks to achieve a specific behaviour in one or more agents by modifying the environment of the agents. However, mechanism design is more concerned with the interactions between agents, and in this paper we will assume that agents do not directly interact.

Policy teaching is also related to applications such as apprenticeship learning. Apprenticeship learning uses the actions of an expert to guide the behaviour of an agent. Abbeel and Ng [1] provided a method of extracting an expert's reward function given the expert's behaviour, and used this to determine the behaviour of an agent. The authors extend this work to solve apprenticeship learning when dynamics (state transition probabilities of the MDP) are unknown [2]. Syed and Schapire [9] modify the work of Abbeel and Ng to provide a method that not only learns from the expert, but also attempts to find a better policy than the expert. However, these apprenticeship learning techniques assume we have complete control over the agent's reward function, while we will be interested in only allowing limited changes to agent's rewards.

The problem examined by Monderer and Tennenholtz [6] does not assume complete control over an agent's reward function as in apprenticeship learning, but does still allow unlimited, non-negative incentives. In k-implementation, an interested party influences the behaviour of agents in a game through the use of incentives. This is closely related to multiagent policy teaching. However, the models of the environment and the incentives are different in k-implementation and our setting. Firstly, k-implementation deals with games of interaction between the agents in a single-shot setting, whereas policy teaching works in a sequential planning setting. More significantly, incentives in k-implementation are provided to agents based on their particular strategy, rather than state-based incentives. These incentives are conceptually in the form of bonus payments made specifically to agents rather than environmental modifications, and as such each agent can receive different incentives.

Eidenbenz et al. [4] discuss how incentives added to a game can be used to manipulate the outcome. An interested party can modify the outcome both to increase or to decrease

the social welfare of the agents. In this paper, however, we assume the interested party has no interest in the actual utility of the agents. The interested party merely assumes the agents will act so as to maximise their utility in the current environment.

3. MULTIAGENT POLICY TEACHING

In *multiagent policy teaching*, the goal of the interested party is to direct a set of agents to a particular policy by adding incentives to the environment. When moving to the multiagent setting, several additional challenges occur that were not originally present in the single-agent approach. Incentives are no longer specifically tailored for a particular agent, but rather for several agents. A key point in the multiagent model is that each agent acts in the same environment and thus has the same incentives. Due to this, the multiagent problem cannot be solved by simply running the single agent policy teaching method separately for each agent.

With direct policy teaching, the IP has a single policy it wishes all the agents to adopt. However, in a multiagent setting, the IP may not be able to teach the target policy to all agents simultaneously. In this case the IP can instead look for the largest subset of agents that can be taught the target policy with a single incentive function. Alternatively, the IP can use a value-based approach, which we discuss in Section 4.

3.1 The Multiagent Model

Before progressing any further, we explicitly outline the model and assumptions used in this paper. The multiagent extension of policy teaching and environment design in this paper deals with *non-interacting agents*. Agents that interact must anticipate the strategies of the other agents and act accordingly. With non-interacting agents, each agent only needs to consider its own actions and the current environment. Without interaction, we can model all agents as planners in MDPs that have the same $M_{-R} = \{S, A, P, \gamma\}$. However, each agent, i, has a different reward function, R_i, and it is assumed that these values are known to the interested party.

The abilities of the interested party remain unchanged from the single agent setting. The IP performs policy teaching though environment design by adding an incentive function Δ to the reward function of each agent. However, this incentive function must be applied to all agents equally, rather than through individual incentive functions. The motivation for this restriction is that, conceptually, the incentives are modifications to the environment rather than additional rewards provided separately by the IP to the agents. Thus, a particular environmental modification affects all agents.

3.2 Teaching All Agents

If there is a single incentive function that can lead all agents to the desired policy, this can be found with a simple linear program. This linear program has the same number of constraints and variables as in the single agent case, regardless of the number of agents.

Let P_a denote the state probability transfer matrix for playing action a in each state. Let P_{π_t} denote the state probability transfer matrix for following policy π_t. That is, $P_a(s_1, s_2)$ is the probability of transferring from state s_1 to s_2 given action a. $P_{\pi_t}(s_1, s_2) = P_a(s_1, s_2)\ \forall s_2$ if $\pi(s_1) = a$. For each agent $i \in N$, by applying inverse reinforcement learning, we obtain a set of up to $|S| \times |A|$ linear constraints. These are calculated as follows, starting with the constraints obtained from IRL [7].

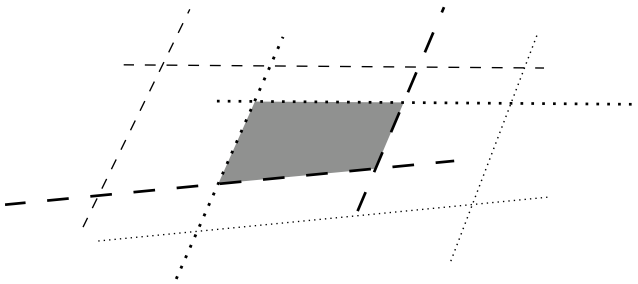


Figure 1: The dotted lines and dashed lines are the constraints for two agents. Taking the most restrictive of each pair of parallel constraints defines the overlapping (shaded) region.

$$\underbrace{(P_{\pi_t} - P_a)(I - \gamma P_{\pi_t})^{-1}}_{C_a}(R_i + \Delta) \succeq \epsilon \ ,$$

$$\begin{aligned}
& && \forall a \in A \setminus \pi_t(s), i \in N \\
\Rightarrow & C_a R_i + C_a \Delta \succeq \epsilon && \forall a \in A \setminus \pi_t(s), i \in N \\
\Rightarrow & C_a \Delta \succeq \underbrace{\epsilon - C_a R_i}_{D_{ia}} && \forall a \in A \setminus \pi_t(s), i \in N \\
\Rightarrow & C_a \Delta \succeq D_{ia} && \forall a \in A \setminus \pi_t(s), i \in N
\end{aligned}$$

Note that the $\epsilon > 0$ is to ensure that π_t is a unique optimal policy. If $\pi_t(s) = a$ then row $(P_{\pi_t}(s) - P_a(s)) = 0 \Rightarrow C_a(s) = 0$. Thus, these rows are removed from the set of constraints. All other rows ensure actions not in the target policy result in expected rewards that are at least ϵ lower than those of the target policy. This makes all π_t actions strictly preferred and so π_t is the unique optimal policy.

If we wish to find an incentive function to induce the desired policy in *all* agents, we must satisfy the $|S| \times |A|$ constraints for *each agent* simultaneously. That is, we must meet all $|S| \times |A| \times |N|$ constraints. However, an observation about the set of constraints allows us to greatly reduce the total number. Let $C_a(s)$ and $D_{ia}(s)$ denote the sth row of C_a and D_{ia} respectively. For any state s and action a the constraint $C_a(s)\Delta \geq D_{ia}(s)$ is parallel to $C_a(s)\Delta \geq D_{ja}(s)$ for every pair of agents $i, j \in N$, as $D_{ia}(s)$ is simply a scalar constant. This observation means that we can run a preprocessing step that reduces the number of contraints to $|S| \times |A|$ by keeping only the most restrictive constraint for each state-action pair (see Figure 1).

If we let

$$\hat{D}_a(s) = \max_{i \in N} D_{ia}(s) \qquad \forall a \in A, s \in S \tag{3}$$

we get the following set of constraints:

$$C_a \Delta \succeq \hat{D}_a \qquad \forall a \in A \setminus \pi_t(s) \tag{4}$$

Now, finding the minimal Δ to induce the desired policy

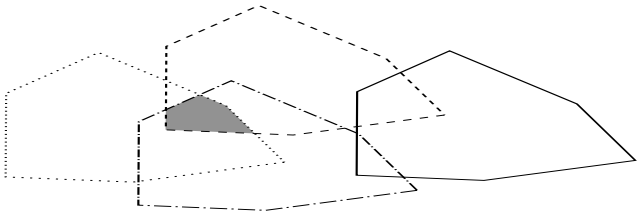


Figure 2: Each polygon represents the space of incentive functions that satisfy all constraints for a particular agent. There is no point that satisfies the constraints of all agents but the shaded region satisfies those of the greatest number of agents.

in all agents can be found with the following linear program:

$$\min_{\Delta} V_{\Delta}^{\pi_t}(start) \tag{5}$$

subject to:

$$C_a\Delta \succeq \hat{D}_a \qquad \forall a \in A \setminus \pi_t(s)$$
$$0 \leq \Delta \leq \Delta_{max}$$
$$V_{\Delta}^{\pi_t}(start) \leq D_{max}$$

The final two constraints ensure the incentives are admissible. Here $V_{\Delta}^{\pi_t}(start)$ is defined as in Equation (2), as the expected discount incentive spending. The linear program has the same number of constraints and variables as the solution for a single agent [11], and this remains constant with respect to the number of agents.

3.3 Determining If All Agents Can Be Taught

As shown in the previous section, we can define the space of possible incentives that induce the target policy in a particular agent. Combining all the constraints gives the space of possible incentives that induce the target policy in all agents. This space is defined by the constraints in Equation (4).

If these constraints, coupled with the admissibility constraints, define an empty region, then no such admissible incentive function exists that can teach all agents the desired policy (as in Figure 2). Linear programming techniques such as the simplex method or extensions to the interior point method will detect the infeasibility of a set of constraints [10]. If, when running the linear program in Equation (5) on the set of constraints for all agents, it declares the problem infeasible, then no incentive can teach the policy to all agents. However, if all agents can be taught, the linear program will return the incentive function that achieves this.

3.4 Teaching The Largest Subset

Given a set of agents, N, where $|N| = n$, the interested party may desire to find the largest subset of N such that all agents follow the desired policy, given an incentive function. Geometrically, let $K = \{K_1, K_2, \ldots, K_n\}$, where K_i is the space of admissible incentives functions that induce the desired policy for agent $i \in N$. The goal is to find the largest subset of K such that the intersection of all elements (e.g. the shaded region in Figure 2) is non-empty.

Given the results in Subsection 3.3, a brute force approach to this problem is to find the largest set of agents that leads to a feasible linear program by testing all possible subsets of agents in order of decreasing cardinality. As soon as a feasible linear program is found based on the constraints of the subset of agents, the algorithm returns the solution to the linear program. If a specific application is likely to only have feasible subsets of size strictly less than $\frac{|N|}{2}$, then the algorithm should test subsets in increasing order of cardinality.

4. MULTIAGENT VALUE-BASED POLICY TEACHING

In the previous section, the interested party was only concerned with agents following the target policy exactly. Where the requirements of the interested party aren't so strict, a value based approach to policy teaching is more appropriate. As an example, consider a simple grid world where an agent's actions allow it to move to adjacent tiles in the grid. The interested party has a desired path for the agents to traverse the grid from the start state to a goal tile. Instead of only accepting solutions where the agents follow the path precisely, a value-based approach permits deviations. The larger the deviation from the target policy or behaviour, the lower the value the IP places on the agent's policy.

In the value-based setting, the IP has a policy valuation function $V : A^S \rightarrow \mathbb{R}$ that assigns a value to each policy. A state-based valuation has a value for each state in the MDP, and calculates a policy value based on the states it visits. An alternative measure is to evaluate each policy based on its *behavioural similarity* to the target policy. In such a setting, the IP needs a function to aggregate these individual policy valuations for the group of agents. While this is not an issue in the single agent setting as there is only one policy to evaluate, it becomes very important in the multiagent case. There are two intuitive methods of determining a total value for a set of policies, motivated by social welfare functions. Where the policy valuation directly represents the utility of the interested party, it makes sense to use a utilitarian approach and sum over all valuations. Thus, the value of a set of policies $\{\pi^i\}$ is calculated as $\sum_{i \in N} V(\pi^i)$. Alternatively, the interested party may prefer to have reasonable values for all agents rather than high values for some and poor values for others. In this case, the egalitarian value of a set of policies is determined by the worst policy, that is $\min_{i \in N} V(\pi^i)$.

4.1 State-Based Policy Valuation

In state-based policy valuation, the value of a policy is calculated as the discounted sum of future rewards, using the reward function of the interested party. This approach was used in the single agent setting by Zhang and Parkes [11]. A multiagent solution is simply a basic extension to the mixed integer program (MIP) used for the single agent problem.

Unlike in direct policy teaching, the additional constraints for multiple agents can not be trivially collapsed together. The MIP needs to calculate the value of each agent's policy, which also requires constraints for each agent to determine this policy. However, as the agents do not interact, the number of constraints and variables only grow linearly with the number of agents. The multiagent MIP has a copy, for each agent, of all the constraints of the single agent solution, except for those bounding the state incentive values, $\Delta(s)$. The minimised function of the MIP aggregates the

policy values for all agents, for example, by summing over all values.

4.2 Comparison-Based Policy Valuation

In some cases, the interested party has a single desired policy or behaviour, but may be satisfied if agents follow a *similar* policy. In such a setting, it is easier for the interested party to appraise policies using some similarity score to the target function than to calculate a state-based reward function that captures this. Unfortunately, there is no clear definition of policy similarity. For policy teaching, we propose that appropriate comparison methods involve looking at the short-, medium-, or long-term behavioural similarities of the policies. In the rest of this section we describe our proposed methods.

Two policies are similar in short-term behaviour if their state transition probabilities are close for most states. This captures similarity in how an agent moves from each particular state and takes into account different actions that have similar state transition probabilities. Stated formally, the short-term difference between policies π_1 and π_2 can be defined as $\sum_{i,j \in S} \left(P_{\pi_1}(i,j) - P_{\pi_2}(i,j)\right)^2$, where $P_\pi(i,j)$ is the probability of transitioning from state i to state j playing action $\pi(i)$. To convert this difference into a policy value based on similarity to a target policy π_t we have:

$$V(\pi) = e^{-\sum_{i,j \in S} \left(P_{\pi_t}(i,j) - P_{\pi}(i,j)\right)^2}$$

Another measure of similarity between policies is based on the medium-term behaviour. In this metric, two policies are similar if they follow similar paths through the state space, or visit states with similar probabilities. An example where such a measure is useful is when an interested party wants the agents to explore a grid world. The desired policy may visit every state with probability at least p after t time steps. There are multiple ways of exploring a grid world, and the interested party is happy with any policy that visits each state with probability close to or greater than p after t time steps. This behaviour is less well defined and thus more difficult to quantify formally.

The long-term behaviour of a policy is measured by the probability distribution over final states as the number of time steps approaches infinity. In this case, the interested party is only concerned with where the agents are likely to end up and not how they get there. A policy's transition matrix P_π defines a Markov chain, and the long term behaviour can be captured by the stationary distribution, r.

Given the stationary distribution for two policies, r_{π_1} and r_{π_2}, the long-term difference between the two policies can be defined as sum of squared difference in stationary distributions: $\sum_{s \in S} |r_{\pi_1}(s) - r_{\pi_2}(s)|^2$. A policy valuation for a policy π compared to a target policy π_t based on long-term behaviour is:

$$V(\pi) = e^{-(r_{\pi_t} - r_\pi)(r_{\pi_t} - r_\pi)^T} \quad (6)$$

For policy teaching, the incentive function is found by maximising some enumeration of policy valuations over all agents. Using the valuation in Equation (6), the maximisation is:

$$\max_{\Delta} \sum_{i \in N} e^{-(r_t - r_i)(r_t - r_i)^T}$$

where r_t is the stationary distribution of the target policy π_t, a constant value, and r_i is the stationary distribution for the optimal policy played by agent i on the MDP with added incentives Δ.

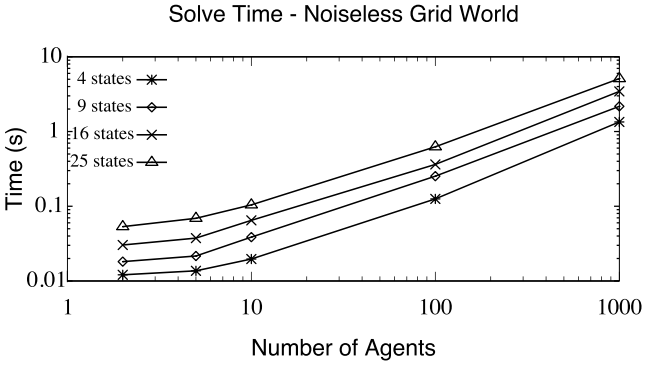


Figure 3: Time taken to find the incentive function to induce a target policy in all agents.

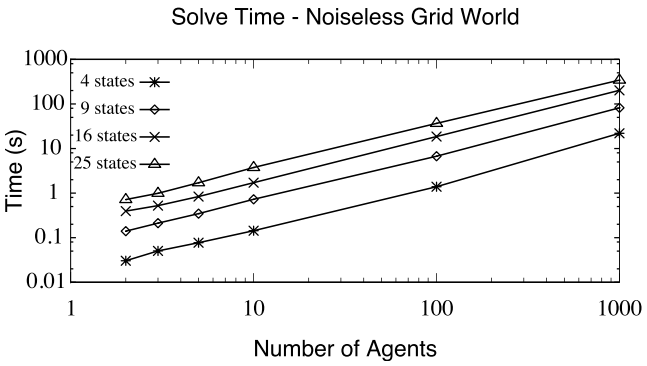


Figure 4: Time taken to find incentives that maximise long-term behavioural similarity of the agents to the target policy.

While this can be converted into a mixed integer program, it requires $O(|S|^2|A||N|)$ constraints and variables. Given that mixed integer programs are typically NP-hard to solve [3], this solution rapidly becomes infeasible. This complexity is further worsened using more sophisticated objective functions that appraise agent policies according to a mix of short-, medium-, and long-term behaviour.

As shown in the next section, good empirical performance is possible by optimising incentives through metaheuristic methods, such as simulated annealing [5].

5. EXPERIMENTAL RESULTS

We conducted empirical tests to verify the solution quality and time requirements of our methods. All tests were run using GNU Octave version 3.0.3 on a 2.4 GHz MacBook Pro with 2 GB of RAM.

The test domain is that of a simple grid world. An agent can either move in one of the four cardinal points or stay put. There is uncertainty added to the environment, where an agent may move in a direction different to the action.

First, we tested the linear program for teaching all agents (Equation (5)). The target policy in these tests was to take the shortest path from the current state to the centre of the grid (with ties broken arbitrarily). The agents' reward functions were generated randomly, with one random "preferred"

state having a value of 1 and all others having a value drawn uniformly from $[0, 0.001]$. The discount factor γ was 0.99. Figure 3 shows the time required to solve the linear program with varying numbers of states and agents. The time required was roughly linear on the number of agents due to calculating the maximum values in Equation (3). The value of D_{max} was chosen such that it was possible to teach all agents the desired policy. In each run, all agents followed the target policy precisely. The difference in solving time on an MDP where an action always moved in the intended direction and where an action had a 10% chance of moving in an unintended direction was not significant.

We also tested the efficacy of a simulated annealing approach to solving comparison-based multiagent policy teaching, as presented in Subsection 4.2. These tests used the same set up of target policy and rewards as in the linear program tests. However, in this comparison-based approach, we looked at policy similarity based on long-term behaviour (using Equation (6)). A high-value policy should have a high long-term probability of being in the centre state with lower probabilities further from the centre. The plot in Figure 4 shows the time required to find the optimal set of incentives, where the algorithm was halted as soon as the method had found a solution within 0.01% of the optimal value. Here, the optimal value was when all agents follow the target policy precisely, and each had a value of 1. The tests used for Figure 4 aggregated agent policy values through summation, but tests using an "egalitarian" measure showed the same linear trend with number of agents. As the simulated annealing algorithm is non-deterministic, the values plotted are the average over 100 different runs. These results show that, while simulated annealing doesn't have the guaranteed performance of a mixed-integer formulation, it is still able to find good solutions in reasonable time.

6. CONCLUSIONS AND FUTURE WORK

This paper examined multiagent extensions to environment design and policy teaching. These procedures allow an interested party to achieve a desired behaviour in a set of agents through environmental modifications. The environmental modifications are in the form of value-increasing incentives added to each state in the environment.

In the case where a policy is being taught to all agents, the paper provided a linear program to solve this problem. The number of constraints and variables in the linear program does not change as the number of agents increases. The linear program is feasible if and only if the policy can be taught to all agents with a single incentive function.

This paper also examined the problem of maximising the aggregate value of agents' policies, from the perspective of the interested party, in particular, when the value of a policy reflects its similarity to some target policy. Policy similarity is defined as the behavioural similarity of two policies, ranging from short-term to long-term behaviour. A comparison-based approach gives added flexibility to the interested party, as a target policy may not need to be explicitly stated. Using long-term behavioural similarity, the IP can compare policies to a target stationery distribution rather than a target policy. The aggregation of values for each agent's policy can be a summation of values, or the minimum of all values. This problem can be solved with metaheuristic optimisation techniques, as a mixed integer programming solution rapidly becomes infeasible. Empirical tests showed that simulated annealing is one such technique that is effective in solving this problem.

A key assumption in this paper was that of no agent interaction. There is added complexity when agents interact, in modelling the environment, determining policies and finding incentives. In this setting, the MDP model is no longer sufficient, but a stochastic game framework would be one appropriate option to examine. Stochastic games are sufficiently general that a conceptually simple, but perhaps computationally difficult environment design method could be developed by adding the interested party as an extra agent to the game. In this method, the interested party's actions do not affect agents' payoffs in the current stage game, but lead to different sets of stage games with modified agent rewards.

Another assumption that can be relaxed in future work is that of complete knowledge of agent rewards. If rewards are unknown, then the policy teaching methods need to incorporate preference elicitation techniques to narrow the space of possible rewards. A multiagent extension of active indirect elicitation [11] will let the interested party narrow the space of possible agent rewards until enough information is known to determine appropriate incentives.

7. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 1, New York, NY, USA, 2004. ACM.

[2] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 1–8, New York, NY, USA, 2005. ACM.

[3] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mixed-integer programming: A progress report. In *The Sharpest Cut: The Impact of Manfred Padberg and His Work*. SIAM, 2004.

[4] R. Eidenbenz, Y. Oswald, S. Schmid, and R. Wattenhofer. Manipulation in games. *Algorithms and Computation*, pages 365–376, 2007.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[6] D. Monderer and M. Tennenholtz. k-implementation. In *EC '03: Proceedings of the 4th ACM conference on Electronic commerce*, pages 19–28, New York, NY, USA, 2003. ACM.

[7] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

[8] S. Parsons and M. Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254, 2002.

[9] U. Syed and R. Schapire. A game-theoretic approach to apprenticeship learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1449–1456. MIT Press, Cambridge, MA, 2008.

[10] M. J. Todd. Detecting infeasibility in infeasible-interior-point methods for optimization. In

Foundations of Computational Mathematics, Minneapolis 2002, London Mathematical Society Lecture Note Series 312, pages 157–192. University Press, 2004.

[11] H. Zhang and D. C. Parkes. Enabling environment design via active indirect elicitation. In *Proc. Workshop on Preference Handling*, Chicago, IL, July 2008.

[12] H. Zhang and D. C. Parkes. Value-based policy teaching with active indirect elicitation. In *Proc. 23rd National Conference on Artificial Intelligence (AAAI'08)*, Chicago, IL, July 2008.

Flexible Procurement of Services with Uncertain Durations

Sebastian Stein
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
ss2@ecs.soton.ac.uk

Enrico Gerding
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
eg@ecs.soton.ac.uk

Alex C. Rogers
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
acr@ecs.soton.ac.uk

Kate Larson
Cheriton School of Computer Science
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, Canada
klarson@cs.uwaterloo.ca

Nicholas R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
nrj@ecs.soton.ac.uk

ABSTRACT

Emerging service-oriented technologies allow software agents to automatically procure distributed services to complete complex tasks. However, in many application scenarios, service providers demand financial remuneration, execution times are uncertain and consumers have deadlines for their tasks. In this paper, we address these issues by developing a novel approach that dynamically procures multiple, redundant services over time, in order to ensure success by the deadline. Specifically, we first present an algorithm for finding optimal procurement solutions, as well as a heuristic algorithm that achieves over 99% of the optimal and is capable of handling thousands of providers. Using experiments, we show that these algorithms achieve an improvement of up to 130% over current strategies that procure only single services. Finally, we consider settings where service costs are not known to the consumer, and introduce several mechanisms that incentivise providers to reveal their costs truthfully and that still achieve up to 95% efficiency.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, multiagent systems*

General Terms

Algorithms, Economics, Reliability

Keywords

Service-oriented computing, service procurement, mechanism design, optimisation

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

1. INTRODUCTION

Increasingly, participants in large distributed systems are able to discover and automatically procure the services of others. This allows service consumers to complete complex computational tasks on demand, but without the need to invest in and maintain expensive hardware. Already, such a service-oriented approach is gaining popularity in a large range of application areas, including Grids, peer-to-peer systems, cloud and utility computing [7].

Despite its benefits, flexible service procurement poses new challenges that have not been addressed satisfactorily by current research. In particular, services offered by external providers are beyond the consumer's direct control and may therefore display uncertainty in their behaviour. Thus, the execution time of services can be highly uncertain, due to concurrent access by other consumers, hardware or network problems and the provider's scheduling policies. This is particularly problematic when services take a long time to complete, as is common for many computationally-intensive tasks, and when consumers need to obtain their results by a certain deadline. Furthermore, in large systems, many different providers may offer functionally equivalent services that are heterogeneous in their quality and costs. This requires consumers to make appropriate decisions about which services to procure, balancing the probability of success with the overall cost. It also necessitates the design of appropriate economic mechanisms that incentivise providers to truthfully reveal their private information, such as their costs and the estimated execution time, thus resulting in good procurement decisions and removing the need for strategic behaviour.

Related to this work is the literature on task allocation under execution uncertainty such as [6]. Here, researchers have studied problems where providers have private information about both their costs for executing tasks, as well as the probability that they will successfully complete their tasks. However, this and similar works do not consider redundancy to increase the overall success probability of task. Now, there is some work that employs redundancy, combining several unreliable services to achieve a higher probability

of success. This includes deployed systems, such as Google's MapReduce [2], but the techniques used for determining how many services to procure are typically ad hoc, and they also do not consider costs. A decision-theoretic approach for addressing the latter is described in [8], but this work assumes that costs are known and focusses on heuristic techniques for complex workflow scenarios. A slightly different approach is taken by work on restarting Web queries, which examines when such queries should be timed out and re-issued (possibly to a different provider) to ensure timely completion [1, 4]. However, such work typically assumes that only one query is active at any time and the costs of multiple queries are not explicitly balanced with the resulting benefit.

To address these shortcomings, we present an abstract model of a procurement scenario with service execution uncertainty. We begin by outlining a generic approach for finding an optimal procurement strategy when service costs and duration distributions are known. This approach is the first to employ redundancy in a flexible and optimal manner as to balance the probability of completing within its deadline and the costs for doing so. More specifically, our approach allows a consumer to invoke multiple services in parallel for executing the same task, and it can dynamically procure further services during execution as the deadline draws closer. To find the optimal strategy, we combine analytical expressions with computational search methods. As brute force is computationally intractable, we present a novel branch-and-bound algorithm that reduces the search, on average, by over 99.9%. We also discuss a heuristic algorithm capable of handling problems with thousands of heterogeneous service providers, and we show that its solutions are, on average, within 0.12% of the optimal. For a range of settings we then experimentally demonstrate that dynamic redundancy achieves an improvement of up to 130% over current approaches.

Next, we examine settings where service costs are private and known only by the providers. For this scenario, we propose a VCG-like mechanism that is incentive-compatible, i.e., that incentivises rational, self-interested participants to reveal their true costs. In this context, it is the first such mechanism that allows consumers to procure multiple, redundant services to increase its probability of success. Moreover, we show that this mechanism can achieve an average 95% efficiency when some prior information about service cost distributions is known to the consumer. To address settings where this is not available, we propose two further novel mechanisms, which have lower information requirements, but still achieve an average 86% efficiency.

In the remainder of this paper, we first present the procurement problem (Section 2) and discuss its optimal solution (Section 3). This is followed by our mechanisms (Section 4) and an empirical evaluation (Section 5). Section 6 concludes.

2. PROBLEM SPECIFICATION

We consider a single service consumer A, which needs to complete a task T. The consumer derives a utility $V \in \mathbb{R}^+$ if the task is successfully completed within a given deadline $D \in \mathbb{R}^+$, and 0 otherwise. Furthermore, there are m service providers, given by the set $M = \{1, \ldots, m\}$, which can complete the task on the consumer's behalf. A consumer can invoke a provider $i \in M$ at any time in the interval $[0, D]$. We assume that, once invoked, the provider remains committed to the task until it is completed (possibly beyond the deadline), and incurs an (expected) cost c_i, where this cost may represent both the running costs of its computational resources and opportunity costs from not being able to use these resources for other tasks. To compensate for these costs, the consumer pays the provider a transfer $\tau_i \in \mathbb{R}^+$ on invocation of i, which is paid regardless of whether the task is completed by the deadline D. Although a provider will always successfully complete the task, the execution time is uncertain, and is given by a continuous cumulative distribution function $F_i(t)$. This denotes the probability that provider i completes the task within time t, and we assume this includes any time needed for pre-/post-processing, queueing and data transfers. We also assume that the execution times of different service providers are independently drawn.

Although we only consider a single task in this paper, crucially we allow multiple providers to execute it concurrently and independently. In this case, the task is considered successful if at least one provider completes it by time D. We assume that all participants are expected utility maximisers.

Now, the key problem is to find an optimal procurement strategy that determines which providers should be invoked and when, such that the consumer's utility is maximised. We compactly represent such a strategy as a vector $\rho = ((s_1, t_1), \ldots, (s_n, t_n))$ with $n \leq m$, where each element represents the invocation time $t_i \in [0, D]$ of a provider $s_i \in M$. A provider i is then only invoked at time t_i (and only receives τ_i) if no provider has so far completed the task. Without loss of generality, we assume that $t_i \leq t_{i+1}$, and $s_i \neq s_j$ if $i \neq j$. For example, assume there are four providers and $\rho = ((2, 0), (3, 0), (1, 2.5))$, i.e., providers 2 and 3 are invoked immediately. Then, if the task has not been completed by $t = 2.5$, provider 1 is also invoked, causing the three providers to run concurrently. Provider 4 is never invoked.

Given a strategy ρ, the consumer's expected utility is:

$$\begin{aligned} U_A(\rho) &= V \cdot \left(1 - \prod_{i=1}^{n} (1 - F_{s_i}(D - t_i))\right) \\ &\quad - \sum_{i=1}^{n} \left(\tau_{s_i} \cdot \prod_{j=1}^{i-1} (1 - F_{s_j}(t_i - t_j))\right). \end{aligned} \quad (1)$$

Furthermore, the expected utility of each provider s_i is:

$$U_{s_i}(\rho) = (\tau_{s_i} - c_{s_i}) \cdot \prod_{j=1}^{i-1} (1 - F_{s_j}(t_i - t_j)), \quad (2)$$

if s_i is included in ρ, and zero otherwise. Furthermore, although our main concern is maximising the customer's utility, as a measure of how well the available services are utilised, we define the overall *efficiency*, also referred to as the social welfare, of a procurement strategy as:

$$\begin{aligned} U(\rho) &= U_A(\rho) + \sum_{i=1}^{n} U_{s_i}(\rho) \\ &= V \cdot \left(1 - \prod_{i=1}^{n} (1 - F_{s_i}(D - t_i))\right) \\ &\quad - \sum_{i=1}^{n} \left(c_{s_i} \cdot \prod_{j=1}^{i-1} (1 - F_{s_j}(t_i - t_j))\right). \end{aligned} \quad (3)$$

This measures the overall quality of a strategy for all participants and therefore ignores any transfers, as these only re-distribute utility between the agents.

3. OPTIMAL SERVICE PROCUREMENT

We now consider the problem of finding the optimal procurement strategy that maximises the consumer's expected utility, given that the consumer has full information about both the providers' costs c_i and the duration distributions F_i. This corresponds to a service-oriented system where providers advertise their services at a fixed price, and thus c_i denotes the advertised price. In this case, we set the transfers to the providers so that they equal these prices, i.e., $\tau_i = c_i$.

More formally, let $\rho^* = \text{argmax}_\rho \, U_A(\rho)$. Finding the optimum, ρ^*, is non-trivial since it involves selecting an appropriate subset of providers, ordering them and then determining invocation times. To solve this, we initially assume that the optimal subset of providers and their ordering is given. That is, we are given an ordered set of providers $\rho^*_s = (s_1, \ldots, s_n)$ where s_i is invoked before s_{i+1}. To compute the optimal procurement schedule, we must determine $\rho^*_t = (t_1, \ldots, t_n)$, where t_i is the invocation time of s_i. To this end, we compute the gradient of the expected welfare, $\nabla U(\rho^*_t)$, and find its root, i.e., $\nabla U(\rho^*_t) = \mathbf{0}$. This results in a system of n simultaneous equations, with one equation for each t_i, with constraints, $\forall i : 0 \leq t_i \leq D$, and $\forall i, j : i \leq j \leftrightarrow t_i \leq t_j$. Solving these equations depends on the family of duration distributions and can be done either analytically or numerically using standard optimisation software. In what follows, we focus on the exponential distribution as this is commonly used for modelling uncertain service durations [9].

3.1 Exponentially Distributed Durations

We now derive analytical expressions for the invocation times ρ^*_t, given ρ^*_s and given that the duration distributions of providers $i \in M$ are given by $F_i(t) = 1 - e^{-\lambda_i t}$, where $\lambda_i > 0$ is a rate parameter. Re-writing Equation 1 with these distributions, and computing the gradient, allows us to compute the optimal invocation time t_i of provider s_i by solving:

$$0 = -V \cdot \lambda_{s_i} \prod_{j=1}^{n} e^{-\lambda_{s_j} D} \prod_{j=i+1}^{n} e^{\lambda_{s_j} t_j} + c_i \sum_{j=1}^{i-1} \lambda_{s_j} \prod_{k=1}^{i} e^{-\lambda_{s_k} t_i} - \lambda_{s_i} \sum_{j=i+1}^{m} \left(c_{s_j} \prod_{k=1}^{j-1} e^{-\lambda_{s_k} t_j} \prod_{k=i+1}^{j-1} e^{\lambda_{s_k} t_k} \right) \quad (4)$$

Here, we note that t_i is independent of any $t_j, j < i$, i.e., the invocation time of a provider does not depend on the invocation time of those already running. This is a result of the exponential function being memoryless, i.e., the probability of completing the task within the next time interval Δt is independent of when it was invoked. Hence, we can calculate each t_i by backward induction, starting with the last provider, n. The invocation time of this can be obtained directly by taking the derivative with respect to t_n (as in Equation 4):

$$t_n = \frac{\ln\left(V \cdot \lambda_{s_n}\right) - \ln\left(c_{s_n} \cdot \sum_{j=1}^{n-1} \lambda_{s_j}\right) - D \cdot \sum_{j=1}^{n} \lambda_{s_j}}{\sum_{j=1}^{n} \lambda_{s_j}} \quad (5)$$

Furthermore, we can obtain a simpler closed-form solution for the remaining invocation times by combining and manipulating the partial derivatives for t_i and t_{i+1}, resulting in:

$$\frac{c_{s_i}}{\lambda_{s_i}} \sum_{j=1}^{i-1} \lambda_{s_j} \prod_{k=1}^{i-1} e^{-\lambda_{s_k}(t_i - t_k)} - \sum_{j=i+1}^{m} \left(c_{s_j} \prod_{k=1}^{j-1} e^{-\lambda_{s_k}(t_j - t_k)} \right)$$
$$= \frac{c_{s_{i+1}}}{\lambda_{s_{i+1}}} \sum_{j=1}^{i} \lambda_{s_j} \prod_{k=1}^{i} e^{-\lambda_{s_k}(t_{i+1} - t_k)} - \sum_{j=i+2}^{m} \left(c_{s_j} \prod_{k=1}^{j-1} e^{-\lambda_{s_k}(t_j - t_k)} \right)$$

Then, using algebraic manipulations, we isolate t_i, and derive an expression that is based solely on t_{i+1}:

$$t_i = t_{i+1} - \frac{1}{\sum_{j=1}^{i} \lambda_{s_j}} \ln\left(\frac{c_{s_{i+1}} \lambda_{s_i} \sum_{j=1}^{i+1} \lambda_{s_j}}{c_{s_i} \lambda_{s_{i+1}} \sum_{j=1}^{i-1} \lambda_{s_j}} \right) \quad (6)$$

Note that Equation 6 is not well defined for t_1, and the optimal here is to set $t_1 = 0$. This is because the cost will be incurred in any case and any delays would only reduce its probability of success by the deadline. Furthermore, we note that the equations can yield negative values for some t_i, indicating that the optimal values lie outside the constraints of the problem (i.e., before the task can be started). In this case, as t_i does not influence the procurement times of later providers, the optimal choice is to set $t_i = 0$, i.e., the provider is invoked at the earliest possible time. Finally, the equations can sometimes yield inconsistent values, i.e., $t_i > D$ or $t_i > t_{i+1}$ for some i, but this only occurs when the ordering and/or the set of providers was non-optimal in the first place.

So far, Equations 5 and 6 allow us to efficiently calculate the optimal procurement times for a given, optimal ordered sequence of service providers ρ^*_s. However, it is not obvious how to find this order. Related work on economic search, such as [10], does not apply to this case, due to the overlap of concurrently invoked providers. Furthermore, our problem includes a fixed time constraint, by which the task has to be completed. Other greedy approaches that order services by increasing costs, decreasing rate parameters, the ratio of these, or approaches that first select providers who individually yield a higher expected utility, also do not always find optimal solutions. This is because it is often best to select cheaper, slower providers first and only invoke the more expensive and faster ones later, to ensure that the task is completed successfully. However, when the deadline of the task is particularly short, the consumer may be forced to immediately invoke the faster, expensive providers.

As a simple example of this, we consider a set of two providers, $M = \{1, 2\}$. The first is cheap and slow with $c_1 = 0.2$ and $\lambda_1 = 0.1$, while the second is expensive and fast with $c_2 = 5$ and $\lambda_2 = 10$. If we then assume that a consumer has a task T with deadline $D = 1.5$ and utility $V = 100$, the optimal procurement strategy is $\rho^* = ((1, 0), (2, 0.75))$. However, if we decrease the deadline slightly to $D = 1$, the optimal strategy becomes $\rho^* = ((2, 0), (1, 0.84))$, thereby reversing the order of invoked providers.

This observation suggests that a simple greedy search for

Algorithm 1 Branch-And-Bound Algorithm.

1: $\rho_s^* \leftarrow ()$ ▷ Best ordering found so far
2: $u_{\text{lower}} \leftarrow 0$ ▷ Best current lower bound
3: $Q \leftarrow \{\rho_s^*\}$ ▷ Unexpanded orderings
4: **while** $Q \neq \emptyset$ **do** ▷ More unexpanded?
5: $\rho_s \leftarrow \text{argmax}_{\rho_s \in Q}$ LOWER(ρ_s) ▷ Pick best
6: $Q \leftarrow Q \setminus \{\rho_s\}$ ▷ Remove ρ_s from Q
7: $P_s' \leftarrow$EXPAND(ρ_s) ▷ Expand ρ_s
8: $P_s' \leftarrow$FILTERDOMINATED(P_s') ▷ Remove dominated
9: **for all** $\rho_s' \in P_s'$ **do**
10: $\check{u} \leftarrow$LOWER(ρ_s') ▷ Find lower bound
11: $\hat{u} \leftarrow$UPPER(ρ_s') ▷ Find upper bound
12: **if** $\hat{u} > u_{\text{lower}}$ **then** ▷ Sufficient upper bound?
13: **if** $\check{u} > u_{\text{lower}}$ **then** ▷ Better lower bound?
14: $\rho_s^* \leftarrow \rho_s'$ ▷ Keep as current best
15: $u_{\text{lower}} \leftarrow \check{u}$
16: $Q \leftarrow Q \cup \{\rho_s'\}$ ▷ Keep for future expansion
17: $Q \leftarrow \{x \in Q \mid \text{UPPER}(x) > u_{\text{lower}}\}$ ▷ Filter orderings
18: **return** FINDTIMES(ρ_s^*) ▷ Return best strategy

the optimal strategy is insufficient. Hence, in the following sections, we present an optimal branch-and-bound algorithm. As this becomes slow when there are dozens of providers, we also discuss a fast heuristic algorithm.

3.2 The Branch-And-Bound Algorithm

Finding an optimal subset and ordering of providers ρ_s^* using a brute-force search is clearly infeasible when the number of providers rises beyond a handful, as the number of possible orderings for m providers is given by $\sum_{i=0}^{m} \binom{m}{i} \cdot i! = \sum_{i=0}^{m} \frac{m!}{(m-i)!}$. However, it is possible to significantly reduce the number of provider orderings that need to be searched by noting that we can use information about some examined orderings to exclude others. For example, assume we have three providers, and we have just considered the ordering $\rho_s = (2, 1)$. This already promises a high utility, and, in fact, we note it is higher than what could possibly be achieved by invoking provider 3 first (e.g., if $V - c_3$ is low). Hence, we can immediately discard all five orderings starting with 3.

This intuition is generalised in our branch-and-bound technique given by Algorithm 1. In more detail, we begin with an empty ordering $\rho_s^* = ()$ (line 1), and then repeatedly consider any new ordering that can be created by appending a single provider *to the end* of an existing ordering. This is implemented by keeping a set of orderings, Q in line 3, that have not yet been expanded in this manner. During each iteration of the main loop of the algorithm (lines 4–17), we then remove one[1] ordering ρ_s from Q (lines 5 and 6) and expand it. Here, EXPAND in line 7 takes an ordering ρ_s and returns the set of all orderings that can be obtained by appending a single remaining service provider from M to ρ_s. From this set of new orderings, we then remove any that include providers that are dominated by others not currently in the ordering (line 8).[2]

For each new ordering ρ_s', we now find both a lower bound and an upper bound for the expected utility that is achievable by any procurement strategy beginning with the providers in ρ_s' (lines 10 and 11). Finding these allows us to exclude any orderings starting with ρ_s' if the associated upper bound is less than the best lower bound found so far. This pruning and updating of the lower bound is performed in lines 12–16.

We now describe LOWER(ρ_s') and UPPER(ρ_s'). To find the lower bound, we simply restrict ourselves to the providers in ρ_s', and find the optimal times ρ_t' for this ordering and return the associated utility, i.e., $U_A(\text{FINDTIMES}(\rho_s'))$, where FINDTIMES returns the optimal procurement strategy using the Equations from Section 3.1. Calculating an upper bound is less obvious, because we may be able to derive significantly higher utility by invoking further services. To this end, we let M' be the remaining service providers that are not in ρ_s'. If $M' = \emptyset$, then the upper bound is equal to the lower bound discussed above. Otherwise, we create a virtual service provider s_ρ with $c_{s_\rho} = \min_{i \in M'} c_i$ and $\lambda_{s_\rho} = \sum_{i \in M'} \lambda_i$. This is based on the rationale that if any providers from M' are invoked in any order, their cost is bound to be at least c_{s_ρ} and their combined probability of success within any given time interval after invocation will never be higher than when immediately invoking all in parallel. With this reasoning, we obtain a new ordering ρ_s'' by appending s_ρ to ρ_s' and then calculate the upper bound as $U(\text{FINDTIMES}(\rho_s''))$. If that is less than the lower bound, this indicates that it is not possible to achieve a higher utility by invoking further providers, and we can set the upper bound equal to the lower bound.

At the end of each iteration, only unexpanded orderings with an upper bound that is higher than the currently highest lower bound are retained (line 17). This limits the size of Q (which we implemented using a priority queue), and also ensures that it is empty when all necessary orderings have been searched. When this happens, the best ordering and associated optimal times are returned (line 18). This final procurement strategy is optimal, because the algorithm searches all orderings, except for those that are known to have a lower expected utility than those already considered. Hence, the optimal ordering will never be discarded from the search.

However, while significantly reducing the search space in most realistic settings, this algorithm still searches for the optimal solution and may sometimes consider a large proportion of the entire search space. This may be the case, for example, when there are large numbers of highly similar providers and when the value of the task is very large in relation to the service costs. To address such scenarios, we introduce a fast heuristic approach in the following section.

3.3 The Heuristic Algorithm

Although we argued in Section 3.1 that a greedy approach does not generally result in an optimal strategy, it can still achieve good results in practice and is more scalable than exhaustive approaches. Hence, we present such an algorithm that starts with an empty ordering and then greedily adds, removes or switches providers until a local optimum is reached.

In more detail, given a current ordering ρ_s and a set of providers M' that are currently not in ρ_s, the greedy approach picks one of the following three actions, in order to maximise the expected utility of its next ordering: (1) it selects a provider $x \in M'$ and adds it to ρ_s at position

[1] We remove the ordering that promises the highest lower bound on the expected utility. This allows us to quickly increase the best lower bound, thereby pruning the search space more effectively.

[2] A provider i dominates j if and only if $(c_i \leq c_j \wedge \lambda_i > \lambda_j) \vee (c_i < c_j \wedge \lambda_i \geq \lambda_j)$. Clearly, it is suboptimal to invoke j before i.

$i \in \{1, 2, \ldots, n+1\}$ (shifting other providers as necessary), (2) it selects a provider s_i in ρ_s and removes it, or (3) it selects two providers s_i and s_j in ρ_s and swaps their positions. This continues until the algorithm cannot find another better ordering. In this case, the current best is returned.

4. MECHANISMS FOR ELICITING COSTS

Whereas so far we have assumed that the consumer, A, has complete information about both the costs and the duration distributions of the providers, here we consider a setting where the cost information is private and unknown to the consumer. Instead, the consumer has to provide *incentives* so as to induce the providers to reveal this information truthfully. Note that we still assume that the providers' duration distributions are known by the consumer, as this information may be obtained from past and shared experiences, e.g using a trust or reputation system, or simply given by the provider.[3]

4.1 $(k+1)^{th}$ Price Mechanism

Typically, when mechanism design is applied to task allocation problems, the well-known Vickrey-Clarke-Groves (VCG) mechanism is used. Providers are asked to reveal their private information (called their *type*) and in exchange are paid a transfer equal to their marginal contribution to the system. This payment structure provides the correct incentives so that each provider willingly reveals their type truthfully [5].

Unfortunately, however, the VCG mechanism is not applicable in our setting, since it is only suited for situations where the types (i.e. the costs) of the providers are independent. In our domain, this property does not hold, since the cost incurred by a provider depends significantly on which other providers are selected in the procurement strategy. Thus, our problem falls into the class of *interdependent types*, and it is well known from the literature that providers no longer have an incentive to reveal their private information truthfully if the VCG mechanism is used [5]. Moreover, any mechanism which ensures that providers truthfully reveal their costs is inefficient [3].

Our first mechanism is the $(k+1)^{th}$ mechanism which works as follows. First, the consumer, A, announces k, $1 \leq k < m$. Then, each of the m providers reports a cost, $\hat{c}_i$, which may differ from their true cost c_i. We assume that providers are ordered so that $\hat{c}_i \leq \hat{c}_{i+1}$, and we define $K = \{i \mid i \in M \text{ and } \hat{c}_i < \hat{c}_{k+1}\}$. That is, K is the set of k providers with the lowest reported costs. These form the *candidate providers* for the procurement strategy. After finding the set K, the payment value τ_i of each candidate provider is set to $\tau_i = \hat{c}_{k+1}$. Now, the procurement strategy used by A is calculated by finding $\rho' = \text{argmax}_{\rho|s_i \in K} U_A(\rho)$ where only providers in K are (potentially) selected to be part of the strategy. It is important to notice that the payment τ_i for $i \in K$ is *conditional*. That is, a payment or transfer to agent $i \in K$ only occurs if the candidate provider is both selected as part of the procurement strategy ρ', and is subsequently invoked. Otherwise, it receives no payment (and incurs no cost).

[3]To verify these distributions in the latter case, a payment scheme based on *scoring rules* could be used in conjunction with our mechanism, see [11], but we leave a more detailed investigation of this issue for future work.

THEOREM 1. *Let M be the set of service providers, $|M| = m$. For any k such that $1 \leq k < m$, the $(k+1)^{th}$ mechanism is incentive compatible in dominant strategies (i.e., truthtelling is optimal, irrespective of what other agents do) and (ex-post) individually rational (i.e., the providers always receive zero or positive utility).*

Proof Sketch. Since the probability of being invoked and the resulting payment are independent of an agent's report, there is no incentive to overreport the cost. Nor is there an incentive to underreport, since this could only result in a situation where $c_i > \hat{c}_{k+1}$, in which case agent i would make a loss. Individual rationaility holds when $c_i = \hat{c}_i$ since $\tau_i = \hat{c}_{k+1} \geq c_i$ for $i \in K$, and thus $\tau_i - c_i \geq 0$. □

While this mechanism has desirable properties, it also suffers from some key limitations. First, it selects providers based solely on their cost information, ignoring the duration distributions, leading to the possibility that expensive providers with fast completion times are excluded. Second, the parameter k must be announced before providers reveal their costs. To set k optimally requires *a priori* information about the distribution of the costs, and expensive calculations and/or simulations (as done in Section 5). To address this last problem, we now introduce two variations of our mechanism.

4.2 Grouping Mechanisms

We introduce two new mechanisms: *Pairing* and *Halving*. These mechanisms differ from the $(k+1)^{th}$ mechanism in both the provider-selection process and the calculation of the (conditional) payment, τ.

In the *Pairing mechanism*, every provider $i \in M$ reports a cost, $\hat{c}_i$. Then, all providers are *randomly* paired with another provider (if $|M| = m$ is odd, then a single triplet is formed). For each pair, the provider with the lower announced cost is placed in the set K and the conditional payment is set equal to the announced cost of the other provider (in the case of a triplet, the provider with the lowest announced cost is placed in K and the conditional payment is equal to the second lowest announced cost in the triplet). All providers not in K are not selected and therefore receive no payment. This results in $|K| = \lfloor m/2 \rfloor$ and $\rho' = \text{argmax}_{\rho|s_i \in K} U_A(\rho)$ as before.

In the *Halving mechanism* all providers in M announce costs as is done in the Pairing mechanism. Then, $\lfloor m/2 \rfloor$ providers are randomly selected and placed into a set G. All other providers are randomly paired and are then treated identically to those in the Pairing mechanism. For members of G, the provider with the lowest announced cost is placed in K and its conditional payment is equal to the cost of the second lowest announced cost from G, while all other providers are discarded. The procurement strategy is computed as before.

THEOREM 2. *The Pairing and Halving mechanisms are incentive compatible and (ex post) individually rational.*

Proof Sketch. Since the pairs and G are formed independently of the agents' reported costs, the proof follows directly from Theorem 1. □

We note that there are many possible variations of these mechanisms, but all would share some key features. First,

the size of K is solely determined by the *number of providers*, and thus does not rely on the consumer choosing an appropriate value. Second, the mechanisms require no *a priori* information about the cost distributions. Finally, they implement *discriminatory pricing* (i.e., different providers receive different payments), information which is then used to form the optimal procurement strategy (given K). On the other hand, the payments are always based on the higher costs (except in the Halving mechanism), and it is therefore not clear whether these variations offer any real benefits in practice. To this end, we experimentally evaluate them in the next section.

5. EVALUATION

We now evaluate our proposed approaches in a variety of simulated environments, to determine if they provide benefits over existing techniques, and to investigate the cost of incentivising providers to reveal their private information. Throughout this section, we randomly generate each provider i by drawing its cost c_i and duration rate λ_i independently and uniformly at random from $[0, 1]$. To consider a range of settings, tasks have either a low ($V_{\text{low}} = 2$) or a high value ($V_{\text{high}} = 8$) and their deadline is either normal ($D_{\text{normal}} = 2$) or urgent ($D_{\text{urgent}} = 0.5$). Furthermore, we repeat experiments 1000 times and use ANOVA and t-tests to ensure statistical significance at the $p < 0.05$ level. As the associated confidence intervals are small, we omit these here for clarity.

5.1 Optimal Service Procurement

First, we consider environments where providers charge their true costs, i.e., $\tau_k = c_k$, and compare the average utility obtained by the optimal procurement strategy[4] described in Section 3 (*Optimal*) to a strategy that always selects the single provider that individually maximises the consumer's expected utility (*Single*). This latter strategy represents current task allocation approaches that do not include redundancy.

The results are shown in Figure 1. Here, we vary the number of providers in the system and plot the average expected consumer's utility, which is equal to the overall efficiency in this setting, as a proportion of V. Observing these trends, it is obvious that using redundancy can significantly improve the consumer's utility and does so in almost all settings considered. In fact, when averaging over all cases considered, the Optimal approach achieves more than a 35% improvement over the Single approach. In particular, when the deadline is small (D_{urgent}) and the task value high (V_{high}), the Optimal strategy is able to employ high redundancy to ensure the task is completed within the deadline, while the higher task utility justifies the additional investment. For example, when there are 50 providers, the Single approach achieves 35.87% of V, while the Optimal achieves 82.65% — a 130% improvement.

[4]This is found using our branch-and-bound algorithm when there are up to ten providers. We then use the heuristic algorithm to obtain a lower bound for the optimal when there are more providers. However, as we show later, the heuristic obtains near-optimal results.

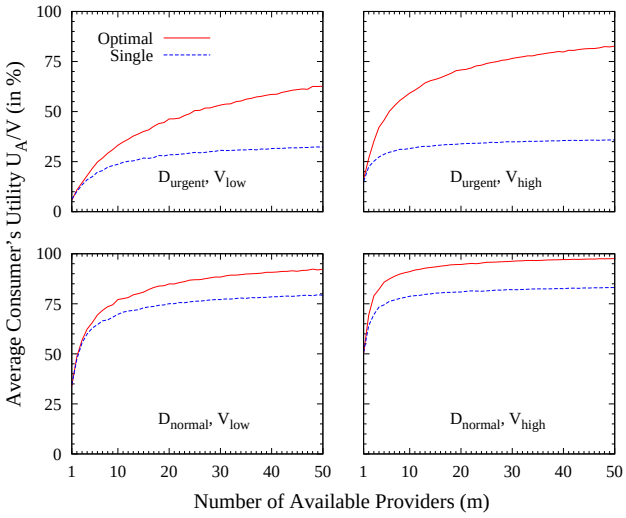


Figure 1: Performance in full information setting.

Next, we note that when solving the above problems, our branch-and-bound approach significantly reduces the computational time required when compared to a brute-force algorithm. For example, when there are 12 providers and we consider V_{high} and D_{urgent}, a brute force approach searches over 1.3 billion provider ordering, which takes an average 3.3 hours (using a Java implementation on a Windows-based Intel 2.2 GHz laptop with 4 GB RAM). By contrast, the branch-and-bound algorithm searches an average 42000 orderings (0.003% of the total search space), finding the optimal in half a second. While the latter still finds solutions for 18–23 providers in minutes (where the brute-force would take over $2 \cdot 10^{10}$ years — longer than the age of the universe), our heuristic approach is better suited for larger settings with hundreds or thousands of providers. To investigate how its performance compares to the optimal, we have applied both to all settings described above with ten or less providers. Over these, the heuristic achieved 99.88% of the optimal.

5.2 Incentive Compatible Mechanisms

Now we consider the mechanisms described in Section 4 and investigate how close the resulting procurement decisions are to the optimal (both in terms of the consumer's utility and the overall efficiency). To this end, Figure 2 compares the performance of a range of $(k+1)^{th}$ price mechanisms with varying k, and our Halving and Pairing mechanisms to the optimal (for brevity, we consider only two representative scenarios here, but similar results are obtained in other settings).

It is immediately obvious here that all mechanisms suffer from a loss in utility for the consumer — a cost that results from incentivising providers to report truthfully. More specifically, the best $(k+1)^{th}$ price mechanism in each case achieves an average 85% of the optimal, while Pairing and Halving both achieve over 70%. We also note that the performance of the $(k+1)^{th}$ depends heavily on the choice of k and can be as low as 25% of the optimal if the wrong parameter is chosen. Furthermore, the best parameter depends on the scenario. For example, for the task with V_{low}, $k = 3$ is the best choice, achieving over 83% of the optimal. However,

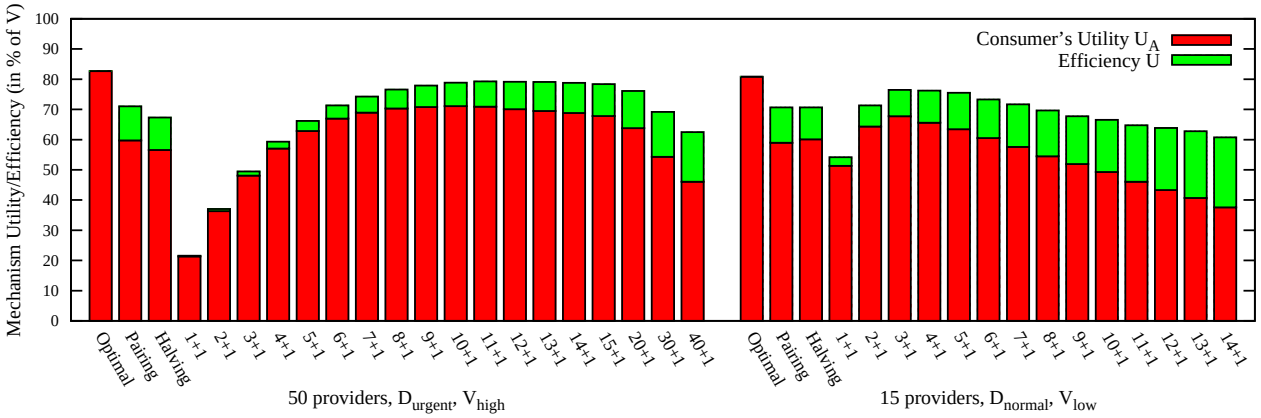


Figure 2: Performance of incentive-compatible mechanisms.

for V_{high}, it is one of the worst, achieving only 58%. Hence, these results indicate that a consumer can achieve a good utility by using appropriate k parameters. However, when insufficient information is available to set k, it can obtain good results by using a Pairing or Halving mechanism.

Next, we consider the overall efficiency, or social welfare. This ignores utility transfers between the consumer and the providers, and therefore gives a better indication of how effectively the available services are used to complete the task at hand. Here, we note that the mechanisms consistently achieve a good overall efficiency. The best $(k+1)^{th}$ mechanism now reaches, on average, over 95% of the optimal efficiency while the Pairing and Halving mechanisms achieve 86% and 84%, respectively.

6. CONCLUSIONS

In this paper, we considered a setting where multiple providers can be redundantly procured to perform a single task which has to be completed within a given deadline. The providers have uncertain execution times, which are given by probability distributions, and incur different costs for executing the task. We first considered the setting with known costs and introduced an algorithm for calculating the optimal procurement strategy, as well as a near-optimal heuristic algorithm for settings with a large number of providers. We then introduced several incentive-compatible mechanisms for eliciting the costs when these are unknown, and we evaluated our approaches empirically. The results showed that redundancy significantly outperforms the standard approach where only a single provider is selected for each task, and it continues to perform well in the incomplete information setting.

In future work, we are interested in investigating a setting where the execution duration distributions are also privately known, and need to be elicited by a consumer. Furthermore, we intend to apply this approach to larger settings with multiple, interdependent tasks.

7. ACKNOWLEDGMENTS

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership (EP/C548051/1). The research was also undertaken as part of the EPSRC (Engineering and Physical Research Council) funded project on Market-Based Control (GR/T10664/01).

8. REFERENCES

[1] P. Chalasani, S. Jha, O. Shehory, and K. Sycara. Query restart strategies for web agents. In *Proc. AGENTS '98*, pages 124–131, 1998.

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[3] P. Jehiel and B. Moldovanu. Efficient design with interdependent valuations. *Econometrica*, 69(5):1237–1259, 2001.

[4] R. M. Lukose and B. A. Huberman. A methodology for managing risk in electronic transactions over the internet. *Netnomics*, 2(1):25–36, 2000.

[5] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[6] R. Porter, A. Ronen, Y. Shoham, and M. Tennenholtz. Fault tolerant mechanism design. *Artificial Intelligence*, 172(15):1783–1799, 2008.

[7] M. P. Singh and M. N. Huhns. *Service-Oriented Computing : Semantics, Processes, Agents*. John Wiley & Sons, Inc., USA, 2005.

[8] S. Stein, N. R. Jennings, and T. R. Payne. Flexible service provisioning with advance agreements. In *Proc. AAMAS08*, pages 249–256, 2008.

[9] K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley & Sons, Inc., USA, 2nd edition, 2001.

[10] M. L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979.

[11] A. Zohar and J. S. Rosenschein. Mechanisms for information elicitation. *Artificial Intelligence*, 172(16–17):1917–1939, 2008.

An Efficient Algorithm For Solving Dynamic Complex DCOP Problems

Sankalp Khanna[1],[2]
[1]Institute for Integrated and Intelligent Systems
[2]The Australian e-Health Research Centre
S.Khanna@griffith.edu.au

Abdul Sattar
Institute for Integrated and Intelligent Systems
Griffith University, QLD 4111. Australia
A.Sattar@griffith.edu.au

David Hansen
The Australian e-Health Research Centre
71/918,RBWH, Herston, QLD 4029. Australia
David.Hansen@csiro.au

Bela Stantic
Institute for Integrated and Intelligent Systems
Griffith University, QLD 4111. Australia
B.Stantic@griffith.edu.au

ABSTRACT

Multi Agent Systems and the Distributed Constraint Optimization Problem (DCOP) formalism offer several asynchronous and optimal algorithms for solving naturally distributed optimization problems efficiently. There has been good application of this technology in addressing real world problems in areas like Sensor Networks and Meeting Scheduling. Most of these algorithms however exploit static tree structures and are thus not well suited to modeling and solving problems in rapidly changing domains. Also, while in theory most DCOP algorithms can be extended to handle complex local sub-problems, we argue that this generally results in making their performance sub-optimal, and thus their application less suitable. In this paper we present new measures that emphasize the interconnectedness between each agent's local and inter-agent sub-problems and use these measures to guide dynamic agent ordering during distributed constraint reasoning. The resulting algorithm, DCDCOP, offers a robust, flexible, and efficient mechanism for modeling and solving dynamic complex problems. Experimental evaluation of the algorithm shows that DCDCOP significantly outperforms ADOPT, the gold standard in search-based DCOP algorithms.

1. INTRODUCTION

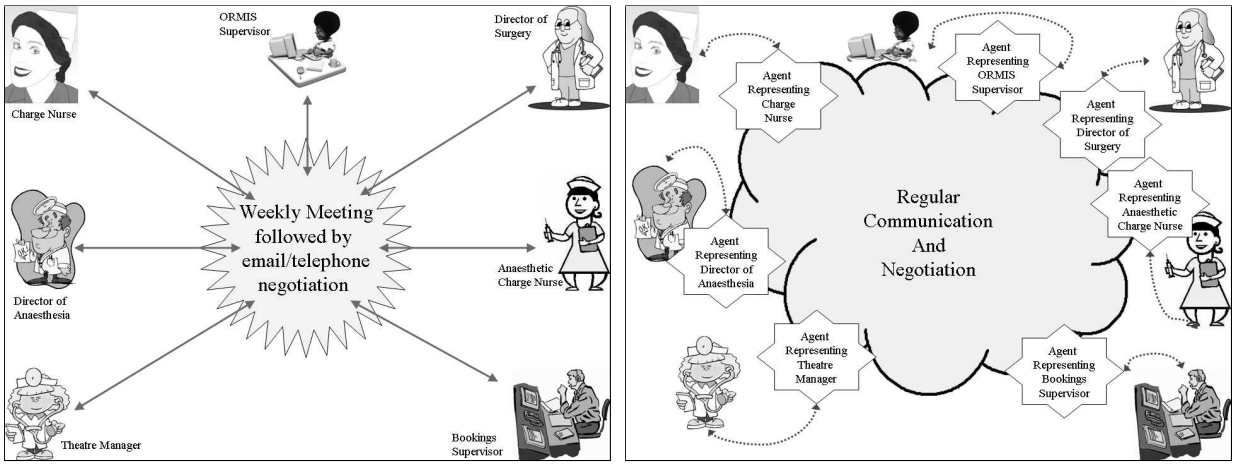
Despite being a relatively young research area, with the first asynchronous Distributed Constraint Satisfaction Problem (DisCSP) algorithm proposed in 1992 [17], and the first complete Distributed Constraint Optimization Problem (DCOP) algorithm, ADOPT, proposed in 2003 [10], the Distributed Constraint Reasoning formalism has developed rapidly to offer efficient and sophisticated algorithms to model and solve a variety of naturally distributed multi-agent problems. Several notable DCOP approaches employing techniques from search (e.g. ADOPT and its several variants), dynamic programming (e.g. DPOP [12] and its several variants) and cooperative mediation (e.g. APO [8]) have emerged and are being successfully used to model and solve problems in sensor networks, meeting scheduling, etc.

This research was motivated by our effort to model and solve naturally distributed complex optimization problems in the health domain, a typical example being the scheduling of elective surgery in a large public hospital. The problem involves several departments, each with its own complex scheduling problem. The departments need to negotiate with each other to build, and maintain, the elective surgery schedule in the face of a dynamic health landscape. Allocation of airport slots to airlines, or public infrastructure to utilities companies, are examples of similar problems in relatively dynamic environments where several agents, each with complex sub-problems, are negotiating in a privacy preserving manner, to optimize a common cost function.

Working towards this aim, the first natural observation is that most current algorithms are based on tree-structures, which are static in nature and would continually need to be rebuilt in dynamic environments. Also, given the nature of the problem domain, partial centralization based strategies would not be a good fit here because of obvious privacy and decision control concerns.

We also note that the metrics used to compare algorithms are questioned by most researchers. Silaghi and Yokoo [16] have shown that it is possible to construct problems that can be exploited by algorithms such as ADOPT and DPOP to exhibit their superiority. Also, Maheswaran et al. [7] show that the performance of ADOPT in solving real world problems is significantly worse than in solving similar-sized map coloring problems.

We draw from Zhou's work [18] in the DisCSP field and gen-



(a) Current Model (b) Proposed Model

Figure 1: Scheduling Elective Surgery at the Princess Alexandra Hospital

eralize the novel measures of constraint density [1], to introduce new DCOP measures of Dynamic Cost Density (DCD) and Degree of Unsatisfaction (DU), and then use these measures to dynamically guide agent ordering in our new Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm. We compare DCDCOP's performance to ADOPT, the current standard in DCOP search, and show that the new algorithm offers a significant improvement over ADOPT.

The rest of this paper is organized as follows. In section 2, we describe our case study of the elective surgery scheduling problem at a large Australian public hospital. Section 3 presents our Multi-Agent System (MAS) architecture for modeling and solving this class of problems. Section 4 follows with a discussion of the DCOP formalism and the shortcomings of the current state of the art in DCOP algorithms in addressing real world dynamic complex problems. In section 5, we present the new measures of DCD and DU, and the DCDCOP algorithm, and explain these with a simple example. Section 6 reports on the empirical evaluation of the algorithm as compared to ADOPT. Lastly, in section 7, we present our main conclusions and discuss ongoing work.

2. SCHEDULING ELECTIVE SURGERY AT THE PRINCESS ALEXANDRA HOSPITAL

Elective surgery is a planned, non-emergency surgical procedure, which can be scheduled at the patient's and surgeon's convenience. The escalating demand for elective surgery is however compounded by a shortage of trained surgeons, anaesthetists and nurses. Recent Queensland statistics show that, as of 1 October 2008, 34, 514 patients were waiting for elective surgery of which almost 21% had waited longer than a clinically desirable time [15].

We conducted an extensive study of scheduling processes followed at the Princess Alexandra Hospital (PAH), a large public hospital in Queensland's capital city of Brisbane. PAH offers 21 operating theaters that can be utilized by various departments for elective surgery. For the process of scheduling, the theater schedule is divided into AM and PM slots of 3.5 hours each. These slots are allocated to various doctors or departments but can be utilized for trauma or emergency if urgently required.

Each department connected (i.e. allocating staff or other resources) to the surgery carries out their individual scheduling activity. The bookings department books patients into the *Bookings Schedule* in consultation with the relevant surgical teams, and records these bookings into the *Operating Room Management Information System* (ORMIS). The different departments can access this information by looking into ORMIS or accessing the latest *Bookings Schedule* on the shared drive, where it is updated everyday at 3PM.

Every Thursday, the managers of the different departments meet and review bookings for the week ahead (Figure 1). Each session is discussed and conflicts in the departmental schedules are worked out by negotiation. Unexpected emergencies, variation in patients' health state, sudden perturbations in staffing, and surgeon availability etc. lead to further changes being often required. However, all changes made to the system after this meeting are dealt with individually by the departments, and resolved on a case-by-case basis using conventional communication such as telephone and emails, or even by face-to-face meetings. In keeping with the dynamics of the domain, the schedule needs to be updated quickly and efficiently. This is often not possible, because of delays in inter-departmental communication, and this leads to the adoption of an easy but inefficient solution resulting in a compromised schedule. For example, if a procedure is canceled at the last minute, because new medical reports say it is no longer required, the bookings department would want to offer the slot to another procedure. They may however be unable to confirm availability of specialist staff or equipment, because the charge nurse or theater manager were temporarily unavailable, which would lead to the slot being unused.

[1]Not to be confused with the traditional measure, i.e. ratio of actual number of constraints to possible number of constraints.

3. PROPOSED MAS ARCHITECTURE

Given the naturally distributed nature of problems like elective surgery scheduling, an intelligent agent based approach is a promising paradigm for modeling the environment.

We propose a methodology where intelligent agents, trained with the constraints, preferences, priorities etc. of the administrators, optimize schedule for their respective departments (Figure 1). They then negotiate in a privacy-preserving manner (i.e. without sharing more information than is essential) to resolve inter-agent constraints. The architecture of each agent (see Figure 2) incorporates an interface module to handle internal and external communication, an intelligence module to handle decision making and learning, and a DCOP engine to drive the optimization.

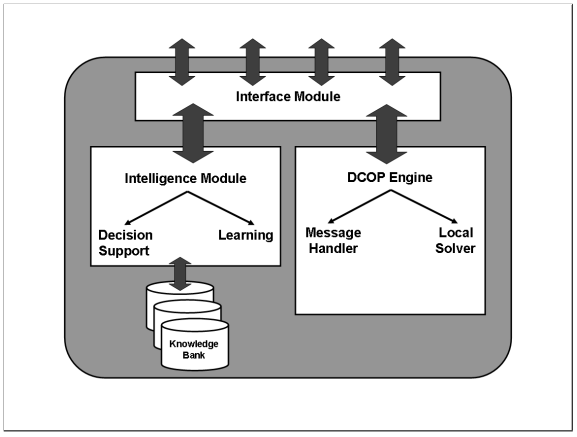


Figure 2: Proposed Architecture for Agent

The agents thus incorporate learning from domain-expert interaction, and have the ability to use intelligent reasoning to translate environmental changes and negotiation requests to constraints for the DCOP engine. Using an appropriate DCOP algorithm the agents optimize their local solution, and then collaborate with each other to resolve inter-agent constraints and align themselves.

The DCOP algorithm thus needs to be robust enough to handle the complexity of each agent's sub-problems and perform inter-agent negotiation in a truly distributed manner, efficient enough to solve the problem in a timely and optimal manner, and flexible enough to adapt to the dynamics of the environment. Additionally if it can shield the local solver mechanism from the inter-agent negotiation process, each agent will have the ability to use a local solver strategy to suit its own need.

4. RELATED WORK

We start with discussing the DCOP formalism and some of the current approaches to solving DCOPs. We then revisit real world dynamic complex problems and discuss the shortcomings of current approaches in addressing these.

4.1 The DCOP formalism

In solving a DCOP, the goal for each agent is to assign values to its variables such that a given global objective function is minimized. The cost functions in DCOP are analogous to constraints in DisCSP, and DCOP is thus regarded as a generalization of the DisCSP formalism. For simplicity, we use the term constraints and cost functions interchangeably. Formally, we can define a DCOP as consisting of:

1. A finite ordered set of Agents $A = \{A_1, A_2, A_3, ..., A_n | n \in Z^+\}$, where, for each Agent A there exists :
   (a) A finite ordered set of variables $V = \{V_1, V_2, ..., V_n | n \in Z^+\}$,
   (b) A domain set $D = \{D_1, D_2, ..., D_n\}$, containing a finite and discrete domain D_i for each variable V_i,
   (c) A constraint set $C = \{C_1, C_2, ..., C_m\}, m \in Z^+$, where each $C_j, \forall j \in [1, m]$ is defined as a cost function on a pair of variables, $f_{i,i'} : D_i D_{i'} \rightarrow N, \forall V_i, V_{i'} \in V$, and
   (d) An ordered solution set $S = \{v_1, v_2, v_3, ..., v_n | v_i \in D_i, \forall i \in [1, n]\}$ where the aggregate cost $F(A) = \sum_{(x_i, x_{i'} \in V)} f_{ii'}(d_i, d_{i'}), x_i \leftarrow d_i, x_{i'} \leftarrow d_{i'} \in A$.
2. The solution set of the DCOP $S^\star$ is defined as the set of the solution sets of each agent.

In keeping with the norm, all constraints are assumed to be binary, and optimization functions are assumed to be associative, commutative, and monotonic [10]. In dealing with complex DCOPs however, we do not assume one variable per agent.

4.2 Current State of the Art

Several DCOP algorithms and their variants have been introduced by recent research. Due to space constraints, we focus on the following key complete optimal algorithms, each representing a significantly different approach. Also, we do not critique each variant as they all still have the same shortcomings when applied to dynamic complex problems.

Asynchronous Distributed OPTimization, or ADOPT [10], is a complete and asynchronous DCOP algorithm. In ADOPT, agents are first prioritized into a Depth First Search (DFS) tree, whereby each agent maintains lower and upper bounds for the subtree rooted at their node. The agents then use opportunistic best-first search to assign their variables such that the lower bound is minimized. *Cost* messages propagate up the tree and *threshold* and *value* messages are sent down the tree, iteratively tightening the lower and upper bounds until the lower bound of the minimum cost solution is equal to its upper bound. If an agent detects this condition, and its parent has terminated, then an optimal solution is found and it may also terminate. The other key idea in ADOPT is to store lower bounds as a threshold and discard partial solutions before they are proven to be definitely suboptimal, thus maintaining linear space complexity at each agent. In the worst case, ADOPT may require an exponential number of messages to arrive at a solution.

Distributed Pseudotree Optimization Procedure, or DPOP [12], is a complete dynamic programming algorithm that involves a three phase process. Similar to ADOPT, the first

phase involves the formation of the DFS tree. Phase two involves calculating and propagating the utility (cost) bottom-up, i.e. from the leaves upwards to the root. Phase three involves a downward value propagation, initiated by the root node. Each agent then calculates its optimal value based on the utility message received from its subtree and the value message received from its parent. DPOP thus generates only a linear number of messages, but the message size grows with every traversal up the tree and the algorithm thus requires large amounts of memory, up to space exponential in the induced width of the problem.

Optimal Asynchronous Partial Overlay (OptAPO) [8] is an alternative approach to DCOP that utilizes partial centralization to solve difficult portions of a DCOP problem. While partial centralization offers an excellent mechanism to solve DCOPs in several scenarios, it is generally unsuitable in the kind of problems we seek to address. In most such cases, negotiating agents would normally refuse to share information or relinquish decision making control of their private sub-problems.

4.3 Solving Dynamic Complex Problems

Since both ADOPT and DPOP utilize static DFS tree structures, changes to the constraints would often result in the need for the tree to be rebuilt. Also, since ADOPT discards no-goods to maintain linear space complexity, changes to the constraints would result in bounds being discarded and the search restarted.

Both ADOPT and DPOP also offer variants to deal with dynamic environments. Modi [9] offers a formalism for mapping and solving dynamic resource allocation problems but this is applied in the DisCSP domain. This is extended to map over-constrained problems into DCOP but can handle only static problems as the author concedes to the lack of an effective DCOP algorithm for dynamic problems. Petcu et al. [11][13] propose S-DPOP and RS-DPOP, which utilize self stabilizing DFS trees to guarantees optimal solution stability in distributed continuous-time combinatorial optimization problems. Lass et al. [6] present another mechanism to deal with the *complicating factor of dynamism* by wrapping ADOPT in an *Adapter* that receives and handles dynamic event requests.

In dealing with the issue of complex sub-problems, algorithms can theoretically utilize decomposition or compilation. In practice however, decomposition results in failure to exploit the inherent benefit of domain centralization, and also blows the distributed problem size out of proportion. Burke and Brown [2] show that the compilation outperforms decomposition in case of large local sub problems but only small domain size, whereas decomposition is more appropriate when the number of inter-agent constraints and domain size is large but only for small problems. Several ADOPT variants use techniques such as decomposition [9], compilation [4], interleaving [3], and relaxation [1], to name a few, to deal with complex sub-problems. All of the above variants however still suffer from the problem of working off a static tree structure that needs rebuilding from time to time.

Further, applying decomposition to DPOP would result in a significant increase in the message sizes, while compilation would need a novel mechanism of calculating the agent utility for different combinations of local variable assignments. It would however be interesting to evaluate the effect of utilizing our measure of DU as the utility metric on DPOP.

We thus conclude that the DCOP algorithms used in all of the above are optimal only in a static environment, and there is need for a more flexible robust algorithm, which can model the complexity and adapt better to a dynamic environment.

5. THE DCDCOP ALGORITHM

We present the new measures of DCD and DU and a new Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm and explain these measures, and algorithm execution, with a simple example.

5.1 Defining DCD and DU

Zhou [18] offers innovative measures of Dynamic Constraint Density (DCD) and Degree of Unsatisfaction (DU) (a measure of how far an agent's instantiation is from reaching a consistent state) and presents algorithms based on these, which can handle complex and dynamic constraint reasoning problems efficiently. These however are suited only to satisfaction and constraint relaxation approaches as the measure does not take varying constraint costs into account.

We generalize the definitions of Zhou to define the following new static measures of Intra-Agent Cost Density ($IACD$) and Inter-Agent Cost Density (I_ACD):

$$IACD_i = \begin{cases} 0, & \text{if } |intraV_i|=0 \\ \frac{\sum_{j=1}^{|intraC_i|}\left(\delta_m\left(intraC_i^j\right)\right)}{|intraV_i|}, & \text{otherwise} \end{cases} \quad (1)$$

$$I\_ACD_i = \begin{cases} 0, & \text{if } |interV_i|=0 \\ \frac{\sum_{j=1}^{|interC_i|}\left(\delta_m\left(interC_i^j\right)+\sum_{l=1}^{|\kappa C_i^j|}\left(\delta_m\left(intraC_i^l\right)\right)\right)}{|interV_i|}, & \text{otherwise} \end{cases} \quad (2)$$

where $intraC_i$ is the set of intra-agent constraints for agent i, δ_m represents the maximum cost of the constraint, $intraC_i^j$ is the j^{th} intra-agent constraint of agent i, $intraV_i$ is the set of variables constrained by $intraC_i$, $interC_i$ is the set of inter-agent constraints for agent i, $interC_i^j$ is the j^{th} inter-agent constraint of agent i, κC_i^j is the set of intra-agent constraints belonging to i and connected to $interC_i^j$, and $interV_i$ is the set of variables constrained by $interC_i$ and controlled by agent i.

The measure of I_ACD takes into account the interconnectedness of the variables that are attached to an inter-agent constraint. So the higher the cost of intra-agent constraints attached to the variable, the greater the impact of the variable on the cost density. This is quite apt as changing the value of this variable would attract a much higher effort towards optimizing the problem. Also, the measures of $IACD$ and I_ACD both equate to zero if there are no intra-agent or

inter-agent variables connected to constraints respectively. This follows from the general idea that a lower value of cost density would mean the problem is closer to the solution. Thus no variables, and a consequent cost density of 0, would imply that this component of the problem does not need to be solved further. We now define the measure of Static Cost Density (SCD):

Static Cost Density of an agent is defined as the sum of the maximum possible Intra-Agent Cost Density and the maximum possible Inter-Agent Cost Density.

$$SCD_i = IACD_i + I\_ACD_i \tag{3}$$

In equations 1 and 2 we replace δ_m by δ_c, which gives us the current cost of the constraint, to get our dynamic measures of IntraUnsat (IU), and InterUnsat (I_U), which represent the dynamic intra-agent and inter-agent cost densities respectively. We utilize these to calculate the measure of Dynamic Cost Density (DCD).

$$IU_i = \begin{cases} 0, & \text{if } |IACD_i|=0 \\ \frac{\sum_{j=1}^{|intraC_i|}(\delta_c(intraC_i^j))}{|intraV_i|}, & \text{otherwise} \end{cases} \tag{4}$$

$$I\_U_i = \begin{cases} 0, & \text{if } |I\_ACD_i|=0 \\ \frac{\sum_{j=1}^{|interC_i|}\left(\delta_c(interC_i^j)+\sum_{l=1}^{|\kappa C_i^j|}(\delta_c(intraC_i^l))\right)}{|interV_i|}, & \text{otherwise} \end{cases} \tag{5}$$

Dynamic Cost Density of an agent is defined as the sum of the current Intra-Agent Cost Density and the current Inter-Agent Cost Density.

$$DCD_i = IU_i + I\_U_i \tag{6}$$

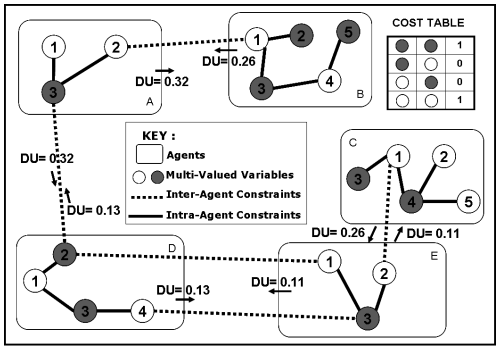
We now redefine the measure of Degree of Unsatisfaction for agent i (DU_i) based on the above :

Degree of Unsatisfaction of an agent is defined as the ratio of the current Dynamic Cost Density to Static Cost Density.

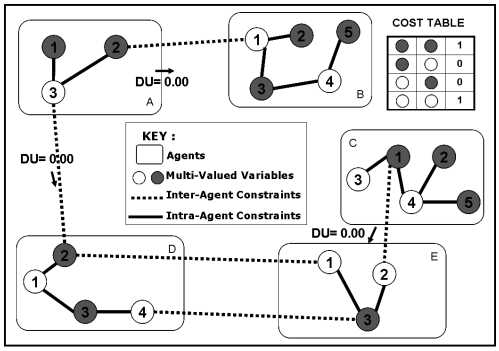
$$DU_i = \frac{DCD_i}{SCD_i} \tag{7}$$

The DU for a problem with $|i|$ agents can similarly be calculated as the ratio of the sum of agent DUs to the number of agents $|i|$. The measure of DU provides a measure of how far away that agent's (or problem's) current instantiation is from reaching a optimal state. It does not provide a direct measure to compare the level of complexity to two agents' problems or the time it may take to solve them. However, unlike a simple summation of max cost or current cost, it attaches a higher cost to the changing of more interconnected constraints, thus providing a more realistic measure.

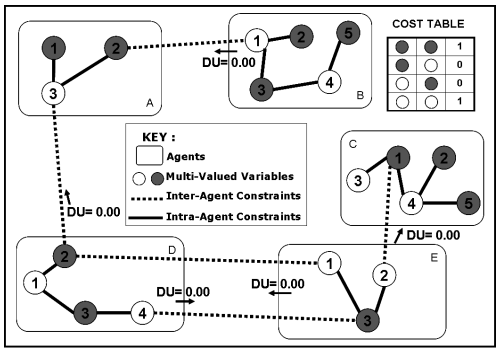
5.2 An Example of Calculating *DU*



(a) Step 1



(b) Step 2



(c) Step 3

Figure 3: Example of DCDCOP Execution

To better understand the above measures, we utilize a simple example as shown in Figure 3(a). Here, we calculate the values of SCD, DCD and DU for agent D, which has four variables, three intra-agent constraints and three inter-agent constraints. The max cost of each constraint (from the cost table shown in the figure) is 1. In calculating the static measures for the problem, we have:

$$\begin{aligned} IACD_D &= \frac{(1+1+1)}{4} = 0.75 \\ I\_ACD_D &= \frac{((1+(1))+(1+(1))+(1+(1)))}{2} = 3 \\ SCD_D &= 0.75+3 = 3.75 \end{aligned}$$

Now, assuming a snapshot view of the scenario, where each

agent has received the communicated values of DU, we can calculate the dynamic measures:

$$\begin{aligned} IU_D &= \frac{(0+0+0)}{4} = 0 \\ I\_U_D &= \frac{((1+(0))+(0+(0))+(0+(0)))}{2} = 0.5 \\ DCD_D &= 0 + 0.5 = 0.5 \\ DU_D &= \frac{0.5}{3.75} = 0.13 \end{aligned}$$

Note that the constraint between variables 1 and 2 of agent D is counted twice in the calculation of I_ACD_D and I_U_D. The calculated value of $DU = 0.13$ will be sent to neighbors A and E.

5.3 Details of Algorithm

The DCDCOP algorithm is implemented as follows :

- All agents start by calculating the values of $IACD$, I_ACD and SCD. Then the agents instantiate their local variables using a Branch and Bound algorithm [5], thus ensuring that the cost at the end of this step is the minimum as per its current context. Each agent then calculates its dynamic measures of DCD and DU and sends its DU and the related context to its neighbors (i.e. those agents with whom it shares an inter-agent constraint).
- All agents then start to receive and handle messages on incoming links. When a message is received, the context of the message is checked to ensure that it is compatible with the agent's $CurrentContext$. If not, the message is discarded and the agent continues to listen on its incoming links. If the message is compatible, the $messageContext$ is added to $CurrentContext$ and the $messageDU$ is compared with the agent's own DU. If the agents own DU is higher, it will reassign its variables, recalculate its dynamic measures and resend messages on outgoing links. If the agent's own DU is lower than $messageDU$, it will not reassign its variables, but if relevant, it will recalculate its dynamic measures and resend messages on outgoing links. In the event that $messageDU = DU$, the agent with a higher static ordering will reassign its variables, recalculate its dynamic measures and send messages on outgoing links.
- The search stops when each agent has achieved a stable state and no more messages are transacted. In the case of a solvable problem, this equates to a situation when the agent, and all its neighbors, arrive at $DU = 0$. In the case of an optimization problem, this equates to a situation when the agent with a higher DU does not change its local solution as doing so would raise the cost of its solution.

The pseudo code of the algorithm is shown in Algorithm 1. As agents negotiate, each negotiation is handled in one of the following ways: if the values of DU are not identical, the agent with a higher value of DU will change its value; if the values of DU are identical, the agents will follow a fixed ordering between them to decide who changes their value. When an agent with a higher DU finds no better instantiation for its local variables, it will return the same, thus reaching a steady state until it receives a context message from other agents causing it to reevaluate its cost function.

Algorithm 1: The DCDCOP Algorithm

```
Calculate static measures
Solve_local_problem
Calculate dynamic measures
Send message (DU, CurrContext) to all neighbors
Receive messages
when received (messageDU, msgContext) do
    if msgContext and CurrContext are consistent then
        add msgContext to CurrContext
        if DU > msgDU then
            Solve_local_problem
        end
        else if DU = msgDU and higher_order then
            Solve_local_problem
        end
        Calculate dynamic measures
        Send message (DU, CurrContext) to all neighbors
    end
end do
Procedure : Solve_local_problem
Branch and Bound to solve local problem
```

Given that we use Branch and Bound, a proven complete algorithm [5], as a local solver, and that inter-agent negotiation results in a stable state, we can intuitively infer that the DCDCOP algorithm is sound and complete. Given the characteristic of the Branch and Bound algorithm, we can also infer that the algorithm exhibits linear space complexity but exponential complexity in time and number of messages in the worst case.

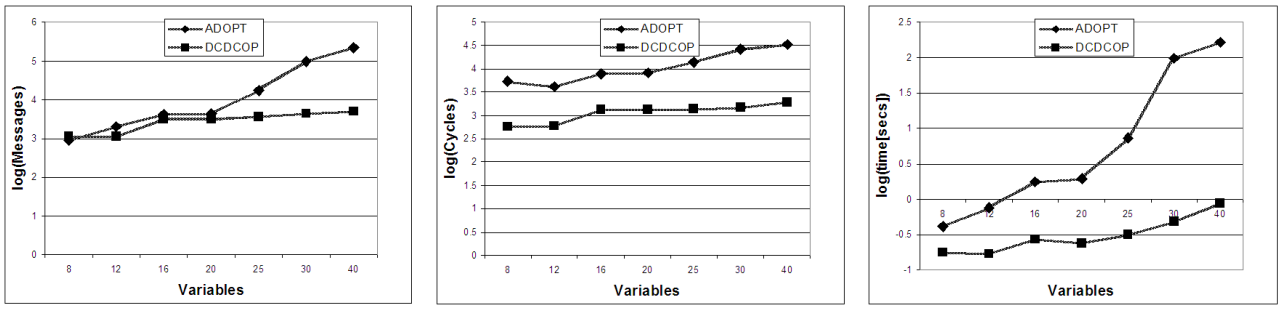
Note that in the case of a dynamic real world problem, the agent with the lower value of DU can force the agent with the higher DU to negotiate by raising the cost attached to the inter-agent constraint.

5.4 Example of DCDCOP Execution

In continuing the example from section 5.2, assuming agents A B and C are first to receive their messages, A and C will both reassign their variables and send out new messages, as they have received DU messages less than theirs. This results in the state shown in Figure 3(b).

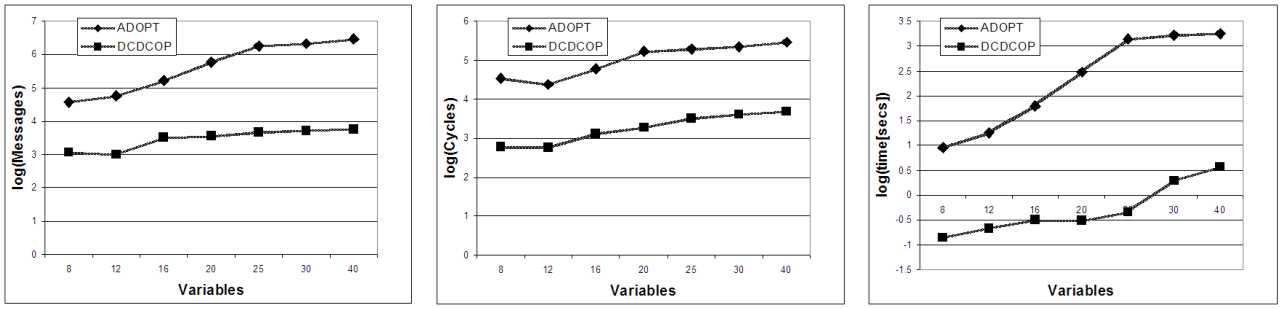
Now assume D and E receive both messages together. Acting on the newer messages, both D and E will attempt to reassign their variables because their last calculated DU is higher than the $messageDU$ from A and C respectively, but with the updated values from A and C, both will arrive at the same local solution as being the lowest cost. Similarly, B will arrive at the same local solution as being the lowest cost and they will send out their DU messages (Figure 3(c)).

At this stage, since all agents will detect that they have a



(a) Log(Number of Messages) (b) Log(Number of Cycles) (c) Log(Time in Seconds)

Figure 4: Performance of ADOPT vs DCDCOP ($LD = 2$)



(a) Log(Number of Messages) (b) Log(Number of Cycles) (c) Log(Time in Seconds)

Figure 5: Performance of ADOPT vs DCDCOP ($LD = 3$)

DU of 0 and their neighbors have a DU of 0, the algorithm will terminate.

6. EXPERIMENTAL EVALUATION

Given the need for agents to negotiate in a privacy preserving manner, otherwise optimal approaches like OptAPO, that use partial centralization, are not suitable in this context. And while DPOP and its variants are good DCOP algorithms, we choose ADOPT for our experimental evaluation because, like DCDCOP, it is based on constraint-guided search and thus provides a more realistic comparison. Further, ADOPT's source code can handle multiple variables per agent without any modification to the algorithm, and is thus well suited to a fair and accurate comparison. To further ensure a reasonable comparison, the new algorithm is developed within the original ADOPT source code [14] and evaluated on the same graph coloring problems that were used to report ADOPT's performance in [9] and come bundled with the source code. Also, ADOPT's original messaging and performance evaluation procedures are utilized for the evaluation.

The original graph colouring problem data (8-40 variables) is evenly distributed between 3-5 agents. Also as in the original evaluation, we analyze the performance of both algorithms on problems with link density (LD) of 2 and 3. The performance is compared using three measures: number of messages, number of concurrent cycles, and time(secs). To prevent a large disparity between the results, the algorithms are run with a maximum time, $timeMax$ of 30 mins. Also, for a meaningful display of results, we report results on a logarithmic scale (base 10) (Figures 4 and 5).

We observe that DCDCOP outperforms ADOPT significantly on all three scales of measurement. The speedup can be attributed partly to the algorithm exploiting domain centralization and performing each local reassignment within one cycle, and partly to the novel dynamic measures used to guide the inter-agent negotiation part of the algorithm.

The results allow for some extremely interesting observations. We observe that the difference in DCDCOP's performance for $LD = 2$ and $LD = 3$ is not significant as in the case of ADOPT. This can be attributed to computational superiority over I/O speeds, the factor responsible for DPOPs performance gains over ADOPT [12]. Further, the performance of DCDCOP with 40 variables (distributed among 5 agents), is reasonably similar to the left side of the ADOPT charts, i.e. 8 agents with 1 variable per agent. Also, the performance of DCDCOP does not deteriorate much as we increase the problem size from 8 variables (in 3 agents) to 40 variables (in 8 agents). This further asserts the computational superiority of the centralized optimization algorithms such as Branch and Bound, and reinforces common belief that communication is the bottleneck in distributed problem solving.

We thus observe how problems that cannot be solved efficiently by DCOP algorithms can benefit greatly if there is a component of domain centralization that they can exploit. Further, a flexible robust algorithm like DCDCOP can help model the departmental centralization structure of

large real world optimization problems, not only offering a natural mapping from real world problem solving structure to DCOP problem solving structure, but also exploiting this to provide an order of magnitude improvement in performance.

7. CONCLUSION AND FUTURE WORK

Several real world optimization problems translate to agents with complex sub-problems in a dynamic environment, that need to negotiate in a manner where privacy and decision making authority is preserved. The current state of the art in DCOP deals with such problems by offering extensions to algorithms best suited for optimization of single-variable agents in a static environment and this often lead to sub-optimality.

We present a flexible, robust algorithm, DCDCOP, that is capable of exploiting the inherent domain centralization found in such problems, and uses a novel measure, DU, to dynamically guide agent ordering during optimization. Experimental evaluation shows that DCDCOP significantly outperforms ADOPT, the current state of the art DCOP search algorithm.

We are currently developing realistic and significant benchmark problems based on scheduling process information and data collected during our extensive interviews and interactions with schedulers and domain experts at the PAH. Further evaluation of DCDCOP and other DCOP algorithms on these benchmarks, and investigating the use of the measure of *DU* to guide agent based negotiation in various DCOP algorithms, are proposed as future work.

8. ACKNOWLEDGMENTS

The authors wish to thank Dr. Peter Moran and his colleagues at the PAH for their ongoing support over the last three years, for allowing us into their world, and sharing their invaluable expertise.

9. REFERENCES

[1] D. Burke and K. Brown. Using relaxations to improve search in distributed constraint optimisation. *Artificial Intelligence Review*, 28(1):35–50, June 2007.

[2] D. A. Burke and K. N. Brown. A comparison of approaches to handling complex local problems in DCOP. In *Sixth International Workshop on Distributed Constraint Reasoning*, page 27–33, Italy, 2006.

[3] D. A. Burke and K. N. Brown. Interleaved search in DCOP for complex agents. In *Proceedings of the Doctoral Program, Principles and Practice of Constraint Programming - CP 2006*, Nantes, France, 2006.

[4] J. Davin and P. J. Modi. Hierarchical variable ordering for distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1433–1435, Hakodate, Japan, 2006.

[5] E. C. Freuder. Partial constraint satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, USA*, page 278–283, 1989.

[6] R. N. Lass, E. A. Sultanik, and W. C. Regli. Dynamic distributed constraint reasoning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1466–1469, Chicago, 2008.

[7] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed Multi-Event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 310–317, New York, 2004.

[8] R. Mailler and V. Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 438–445, New York, 2004.

[9] P. J. Modi. *Distributed Constraint Optimization for Multiagent Systems*. PhD thesis, University of Southern California, USA, 2003.

[10] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 161–168, Melbourne, 2003.

[11] A. Petcu and B. Faltings. S-DPOP: superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI-05*, page 449–454, Pittsburgh, Pennsylvania, USA, 2005.

[12] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland, Aug. 2005.

[13] A. Petcu and B. Faltings. Optimal solution stability in dynamic, distributed constraint optimization. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 321–327. IEEE Computer Society, 2007.

[14] C. P. Portway. USC DCOP Repository. http://teamcore.usc.edu/dcop, 2008.

[15] Queensland Health. Quarterly Public Hospitals Performance Report September Quarter 2008. http://www.health.qld.gov.au/surgical_access, 2008.

[16] M. C. Silaghi and M. Yokoo. DFS Ordering in Nogood-based Asynchronous Distributed Optimization (ADOPT-ng). In *Sixth International Workshop on Distributed Constraint Reasoning*, Italy, 2006.

[17] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.

[18] L. Zhou, J. Thornton, and A. Sattar. Dynamic Agent Ordering in Distributed Constraint Satisfaction Problems. In *Australian Conference on Artificial Intelligence*, pages 427–439, 2003.

Pick-A-Bundle: A Novel Bundling Strategy for Selling Multiple Items within Online Auctions

Ioannis A. Vetsikas, Alex Rogers and Nicholas R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
{iv,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper, we consider the design of an agent that is able to autonomously make optimal bundling decisions when selling multiple heterogeneous items within *existing* online auctions. We show that while bundling the items together into a single lot is effective at reducing listing costs, it also results in a loss in auction revenue. To address this loss we introduce a novel bundling strategy, that we call *pick-a-bundle*, that can be implemented within any existing auction format. We show, mainly using simulations, that this new bundling strategy generates greater expected revenue than the complete bundle of all items, and, by inducing additional competition between bidders, it usually generates greater expected revenue than using separate auctions for each item. In order for our agent to accurately and efficiently calculate its expected revenue when using our new strategy, we derive a novel polynomial time algorithm for calculating the probability distributions of the sum of the top order statistics of i.i.d. variables drawn from any arbitrary distribution. Furthermore, we include in our analysis the strategic behaviour, in terms of bid shading, that the buyers may consider in our new auction format.

1. INTRODUCTION

The bundling of a number of heterogeneous items into a single lot is a common strategy when sellers participate in auctions. For example, within an online auction, such as *eBay.com* or *taobao.com*, sellers may bundle together a small number of low cost items, such as DVDs or computer games, in order to avoid incurring separate listing fees. Likewise, in traditional auctions it is common to find furniture items bundled together into a single lot in order to reduce the time overhead (and cost) involved in selling each item separately.

While the cost saving of bundling items together is self-evident in the examples described above, these savings must be set against the effect that bundling has on the expected revenue of the auction. Specifically, when the items being sold exhibit complementary valuations (i.e. the valuation of the bundle is greater than the sum of the valuations of each individual item) then the rationale for bundling is clear. However, the formal economic literature has little to say regarding the seller's revenue when bundling items that exhibit non-complementary valuations within auctions (as in the example of a bundle of DVDs described above). In particular, research in this area has so far only addressed the problem faced by a multi-product monopolist offering single items or bundles of items at fixed prices [1, 12]. While this setting is somewhat different to the one that we consider, this work shows that the bundling of goods can yield an increase in revenue, even when the items offered exhibit non-complementary valuations. Similarly, more recent work on the bundling of information goods on the internet, again shows profitability even in the absence of network externalities or economies of scale [2, 3]. Jehiel *et al.* [2007] examine a setting with additive valuations, which is similar to ours, however as the authors point out in their conclusions, their results do not work within "standard auctions". This limitation is also present in the computer science literature, where the bundling of items is often studied within the context of combinatorial auctions (see [5] for a review). Furthermore, this work has largely addressed the issue of complementary valuations, and has proposed novel auction protocols that allow bidders to express their preferences for specific bundles of goods [13, 4, 6, 11]. While such results are useful to the designers of new online or real-world auctions, they do not help a seller who is attempting to use an existing auction format (such as the English auctions of eBay or the sealed bid second price auction) in which bidders may only submit bids on the lot offered (and not on subsets of items).

Against this background, in this paper we consider for the first time the effects of bundling non-complementary goods *within a standard auction format*[1]. Our goal is to develop an autonomous *auction agent* that can advise on, and ultimately automate, the process of selling multiple heterogeneous items within such online auctions. To this end, we present a novel bundling strategy, that we call *pick-a-bundle*, in which a seller lists a set of items and announces a bundle size, and the buyers bid for the right to select a number of items from this set, which is equal to the bundle size;[2] the remaining items, which were not selected by the winner in his bundle, are then sold in a second round of separate auctions. For example, five DVDs may be listed and a bundle size of three might be announced, thus the winner of the auction would select the three DVDs that he would like to receive and the remaining two would be sold in a second round of separate auctions afterwards.

The design of our pick-a-bundle auction is informed by the intuition that bundling items will in general reduce the transaction or

[1]We specifically consider a sealed bid second price auction in our analysis, but due to the revenue equivalence theorem, our results apply to any efficient auction protocol.

[2]The idea of choosing the top valued items has also been used in [9, 17]. However, in these settings, the prices are fixed. Furthermore, information goods can be copied, whereas in our setting the fact that the bidders are competing for limited unclonable resources is what creates additional revenue.

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

listing fees of the seller, but by offering just a subset of the items to the buyer, it will also induce competition between bidders who may actually prefer different items. This second factor has been observed in real world auctions that are sometimes used to sell a number of individual apartments within an apartment block. Here, rather than bidding for individual apartments, the buyers bid for the opportunity to select one of the remaining remaining unsold apartments. This procedure generates competition between bidders who prefer different apartments, and results in increased revenue for the seller [Personal communication from Michael H. Rothkopf, 2007]. We show that the same factors influence our pick-a-bundle auction, and show that it generates greater expected revenue than an auction where all items are included in a single bundle, and moreover, in the vast majority of the cases examined, it also generates greater expected revenue than selling each item in separate single-item auctions. Thus, in more detail, this paper makes the following contributions:

- We prove that the standard bundling strategy, in which multiple items are sold together in a single bundle, generates, with few exceptions, less revenue than separate single-item auctions (proposition 1). We address the loss of revenue incurred in this complete bundle auction by describing our novel pick-a-bundle format. Contrary to pre-existing work on bundling, *this format can be implemented within any existing standard auction*, as it does not require a redesign of the auction.[3] We motivate our decision, by showing theoretically that there always exists a pick-a-bundle auction which generates greater revenue than selling the items separately (proposition 2).

- We empirically evaluate our new bundling strategy through simulation, and show that it generates greater expected revenue than separate single-item auctions and the complete bundle (regardless of whether listing costs are taken into account).

- We then address the problem that the aforementioned simulations scale poorly as the the size of the problem (i.e. the number of items and bidders) increases. In order to calculate, rather than simulate, the expected revenue of the pick-a-bundle auction, in the cases where the items are relatively similar,[4] we propose a novel polynomial time algorithm (section 5) for calculating the probability distributions of the sum of the top order statistics of i.i.d. variables drawn from arbitrary discrete distributions.

- Finally, we examine how the presence of the second round of auctions for selling the remaining items, which were not selected by the winner of the pick-a-bundle, affects the bids of the buyers. While in some cases, the bids will not be affected (claim 1), in some others the buyers are going to shade (reduce) their bids. We compute these bids and integrate this effect into our analysis of the pick-a-bundle revenue and our experiments.

The remainder of this paper proceeds as follows: In section 2 we formalize the problem setting. In section 3 we describe and motivate the pick-a-bundle strategy, before discussing how to optimize its performance (by selecting the optimal bundle size) in section 4. In section 5 we present the algorithm for computing the expected revenue without simulations, for the case of similar items. In section 6 we examine the effect of the second round of auctions on the buyers' bids. In section 7 we generate empirical results that, unlike those of section 4, take the effect of bid shading into account; the algorithm of figure 2 is also used to generate the results without the need for simulations. Finally, we conclude.

[3]While we have specifically addressed the needs of an autonomous *auction agent* here, we note that all the results of this paper are also of immediate use to current sellers using existing traditional or online auctions.

[4]We assume that they have the same prior distributions $f_i(u)$ (defined in section 2).

2. PROBLEM STATEMENT

In this section we formally describe the auction setting to be analyzed and the auction format choices that a seller has in order to maximize her revenue. First, however, we introduce the basic notation. We assume that a seller has $m > 1$ items to sell. There are n buyers interested in purchasing some of these items. The valuation u_{ij} of item i to each bidder j is drawn from a known distribution with probability density function (pdf), $f_i(u)$, and associated cumulative density function (cdf), $F_i(u)$. The exact valuations $u_{ij}, \forall i$ are known only to the bidder j himself. These distributions represent the prior knowledge available to all participants about the valuations of the other agents in the auction and are common knowledge to all.[5] We assume that $f_i(u)$ take non-zero values in $[0, L]$, and thus for $\forall u < 0$ and $\forall u > L$, it is $f_i(u) = 0, \forall i$. Although there is nothing to limit the applicability of our experimental results to any distribution, in the experiments we assume that the item valuations and corresponding bids take discrete values. This represents no loss in accuracy since in all real auctions bidders can only place bids that are whole denominations of the currency used (i.e. a bid can't be 95.3 pence, but rather either 95 or 96 pence). Given this, the distributions $f_i(u)$ used, are discrete, and can take values $u = 0, \ldots, L$. In our analysis, we assume that the value of items is additive, meaning that a set of items is worth the sum of the individual item valuations; this is a result of the limited number of results in the literature about how bidders bid in cases of complementary and substitutable items (they don't bid truthfully), which makes it impossible to analyze the expected revenue in such cases.[6]

Sellers sell the items in second price auctions (i.e. auctions in which the top bidder is awarded the item for sale, and the payment is equal to the second highest bid). These auctions are incentive compatible, and thus we assume that bidders place bids equal to their true valuations [10]. Because of the revenue equivalence theorem, our results are also valid in cases that a first price auction would be used (or indeed any other efficient auction mechanism).

Finally, we assume that the seller has a certain degree of freedom when she sets up the auction; even though the auction format must follow that of the online or traditional auction house being used (i.e. in our case it must be a sealed bid second price auction). In particular, the good for sale is determined by the seller, and thus, instead of selling a single item, bundles of items may also be listed.

In real world auctions, there is often a *listing cost* associated with this auction which is levied on the buyer, the seller or both. In our experiments we assume a fixed listing cost C extracted from the

[5]These valuations can model a wide variety of scenarios, simply by selecting the correct distribution. For example, the distribution could incorporate the value of a potential resale by selecting the item valuations (given by the distribution) to be equal to the resale value in the cases when the actual bidder value from keeping the item is lower than the resale value.

[6]On eBay, for example, sometimes a seller will offer to sell substitutable items, like "choose a black or a white iPod". In such cases, the analysis is much easier than in our model, because here bidders are interested in exactly one item; in our model they could be interested in buying both.

seller's revenue for each auction that is conducted. This is inspired from the actual listing costs levied by online auction houses such as eBay, which charge a listing cost consisting of a fixed fee C and a percentage cost (which can be ignored as it reduces the revenue in all cases by the same percentage).[7]

3. THE BUNDLING STRATEGY

A seller with multiple items for sale may choose to do so in *separate auctions each selling a single item*. Another option, which is the main bundling technique used by sellers on sites such as eBay, is to offer a *complete bundle* of items for sale; the winning bidder gets all of the items listed. This is usually done in order to reduce the listing costs. However, as we show below (for the case of identical distributions $f_i(u)$), the complete bundle generates less revenue, except in the case of few bidders:

Proposition 1 *For a symmetric distribution $f(u)$, when the item distributions $f_i(u) = f(u), \forall i$, an auction for a complete bundle of m items, and m separate single-item auctions generate the same expected revenue for the seller when there are $n = 3$ bidders, more for $n = 2$ bidders, and less for every other case ($n > 3$ bidders).*

PROOF (SKETCH). Due to space constraints, we only present here the proof for the case when $n = 3$. The proofs for the other cases (when $n = 2$ and $n > 3$) follow some of the same steps as the part of the proof presented here and also use the observation that taking the convolution of a pdf with itself (any number of times) will concentrate the probability mass towards the mean of the distribution and that the effect is more pronounced as the number of convolutions performed increases. Also note that since f is symmetric, it is $f(u) = f(L-u), \forall u \in [0, L]$, and the cdf is thus $F(u) = 1 - F(L-u), \forall u \in [0, L]$.

Now, when $n = 3$ bidders participate, the expected revenue from each single-item auction, R_{single}, is the expected value of the 2^{nd} order statistic (the second highest valuation) of the valuation that each of the 3 bidders have for that item. This can be computed to be equal to $R_{single} = L - \int_0^L (F^3(u) + 3F^2(u)(1-F(u)))du = \frac{L}{2}$, and can be seen to be equal to the mean of the original symmetric distribution with pdf $f(u)$.

Furthermore, if $g(u)$ and $G(u)$ are the pdf and cdf respectively of the distribution of the sum of m i.i.d. random variables which are drawn from distribution $f(u)$, then, since $f(u)$ is symmetric, we know that $g(u)$ must also be symmetric. Using the same reasoning as in the case of distribution f above, we deduce that the expected revenue from selling the complete bundle R_{bundle} is equal to the mean of distribution $g(u)$, which is equal to $R_{bundle} = \frac{m \cdot L}{2} = m \cdot R_{single}$. Therefore the expected revenue in both cases is the same. □

The complete bundle suffers from a loss in revenue, because the second highest valuation of the bundle is generally less than the sum of the second highest valuations for each separate item, since a bidder's high valuation for one item is likely to be balanced by a lower valuation for another item. We can generalize this observation based on our experimental evidence for any combination of distributions $f_i(u)$ (see sections 4 and 7) and conclude that, while for $n = 2$ the complete bundle is the best option, for $n \geq 4$ bidders participating, one would be better off using separate single-item auctions.

[7]To be more precise, eBay charges an "insertion fee", which is a fixed cost depending on the starting price of the auction and a "final value fee" which is, for the most part, a percentage of the final closing price of the auction.

In order to address this loss in revenue, we propose our *pick-a-bundle* strategy. Under this scheme, the seller advertises all m items that are for sale, but each buyer bids for the right to buy k of these items (where $k \leq m$). The winner of the auction informs the seller which are the k items that it actually wants to receive (i.e. the k items that have the highest valuations for that agent). The remaining items, which are not selected, still need to be sold by the seller, and this is done by having another round of separate single-item auctions.[8] Note that the case when $k = m$ is exactly the complete bundle auction, which is why the complete bundle is a special case of pick-a-bundle. However, when $k = 1$, this is not the same as selling in separate auctions, because there is one auction in which the highest valuation among all agents for all items wins, and then the leftover items are sold in separate auctions.

The intuition behind our bundling strategy is clear. By reducing the size of the bundle compared to the complete bundle, the magnitude of the revenue loss due to the fact that a seller might be primarily interested in only some of the items is reduced. At the same time, the preference of different subsets of items by different bidders also induces additional competition into the auction, as buyers interested in different subsets of items will now compete against each other. This leads our pick-a-bundle auction to outperform both the complete and the separate single-item auctions in the vast majority of settings we examined.

In fact we can further motivate this approach even before we discuss how to evaluate the different auction settings:

Proposition 2 *For any distributions $f_i(u)$, using the pick-a-bundle auction, when $k = 1$ item is chosen, will always generate the same or more revenue than separate auctions each selling one of these items.*

PROOF. Let $u_{i,j}$ be the valuation that bidder i has for the j^{th} item. In the pick-a-bundle auction with bundle size $k = 1$, the revenue R_b is equal to the second highest among the highest values of each bidder. Let i_1 be the bidder with the highest valuation $u_{i,j}, \forall i, j$, for item j_1, and $i_2 \neq i_1$ the bidder with the second highest among the bidders' high valuations; this valuation is for item j_2. Thus $R_b = u_{i_2,j_2}$ and this revenue is obtained for selling object j_1.

The revenue R_{j_1} obtained in selling item j_1 on its own is the second highest value among $u_{i,j_1}, \forall i$. We examine the following two cases:

1. When $j_1 = j_2$. Then $R_{j_1} = u_{i_2,j_1} = u_{i_2,j_2} = R_b$. The revenue in this case is exactly the same.
2. When $j_1 \neq j_2$. We know that $u_{i_2,j_2} = \arg\max_{i \neq i_1,j} u_{i,j}$. Since R_{j_1} is equal to one of the elements of set $\{u_{i,j}\}, \forall i \neq i_1, j$, therefore $R_{j_1} \leq u_{i_2,j_2} = R_b$.

Thus, the pick-a-bundle auction, when one item is chosen (i.e. $k = 1$), will always generate the same or more revenue than any of the individual auctions selling these items. Thus, the revenue from our new auction setting (for $k = 1$) is always at least as high as that from selling the items in separate auctions. □

This shows that, in the case of scenarios where buyer strategic bidding is not an issue (see section 6), the pick-a-bundle auction for bundle size $k = 1$ generates more revenue than separate auctions. This does not mean that selecting $k = 1$ is indeed the best bundle size; in fact quite often it is not. Rather it means that in this setting, it is always preferable to use a pick-a-bundle auction in preference to separate auctions.

[8]Note that given sufficient remaining items, another pick-a-bundle auction could be used on the remaining items. We do not explicitly consider this case, but the analysis that we present could easily be extended to cover it.

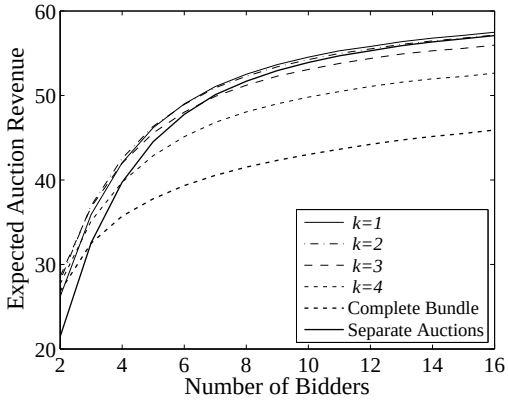


Figure 1: Seller's revenue for a bundle of $m = 5$ items, as the number of participating buyers n increases, when using (i) separate single-items auctions to sell each item individually, (ii) a pick-a-bundle auction initially where $k = 1, \ldots, 5$. The item valuations are drawn from five different, non-independent distributions.

4. OPTIMIZING THE BUNDLE SIZE

Now that we have presented the seller's choices (i.e. separate auctions, complete bundle and pick-a-bundle), she needs to find out which of these choices gives the highest revenue. This involves computing the expected revenue for all these choices, and in the case of the pick-a-bundle auction, also selecting the best value of k (the size of the bundle offered). For the general scenario described in section 2, we need to run a simulation in order to compute the expected revenue of the various cases; we take a large number of random samples of item valuations for all bidders, and average the revenue of the auction across all samples.

To illustrate the wide applicability of our new auction format we carried out a number of simulations with different distributions. Here we show the results for a particular representative case. More specifically, we assume that item valuations u_{ij} are drawn from the following 5 distributions: (i) u_{1j} are drawn uniformly from $\{3, \ldots, 12\}$ (prob. $\frac{1}{10}$ for each value), (ii) u_{2j} are correlated with u_{1j} and are $u_{2j} = u_{1j} + z$, where z is drawn uniformly at random from $\{-2, \ldots, 2\}$, so u_{2j} take values from 1 to 14, (iii) u_{3j} are drawn uniformly from $\{0, \ldots, 14\}$, (iv) u_{4j} are equal to 0 with prob. $\frac{4}{10}$ and take values $\{5, \ldots, 10\}$, each with prob. $\frac{1}{10}$, and (v) u_{5j} are equal to 0 with prob. $\frac{1}{2}$ and equal to 12 with prob. $\frac{1}{2}$.[9]

We present the mean value of the revenue generated in the simulation for all possible cases in figure 1. We repeat the simulation up to 100000 times such that the standard error in the mean values shown is less than the thickness of the line, and hence, we do not show error bars in the figure. As we expected (based in part on the theoretical results of propositions 1 and 2), the complete bundle is not a good option no matter how many bidders participate in the auction. The same is also true for the revenue from separate auctions. The pick-a-bundle auction generates more revenue, no matter whether the listing costs are considered or not.[10] Now, we consider how the choice of the bundle size (i.e. the value of k) affects the revenue of the pick-a-bundle auction, and hence, show how to determine which bundle size is optimal in any given case. For only $n = 2$ bidders, $k = 3$ is the best choice, for a small number of bidders ($n = 3, \ldots, 5$), $k = 2$ is the best choice, whereas for larger numbers ($n \geq 6$), $k = 1$ is the best choice. Depending on the prior belief about possible numbers of bidders participating, n, the auction used should be the pick-a-bundle auction with $k = 1$ or $k = 2$. If the listing cost is included, then $k = 2$ is the best choice for almost all cases ($n \geq 4$), and $k = 3$ is the best choice only when $n \leq 3$.

While these simulations allows us to compute the auction parameters that maximize the seller's expected revenue for any specific setting, they are impractical in cases where the number of bidders, n, or items, m, increases significantly. This is due to the fact that, in this case, each individual simulation run takes longer to compute, and also that each simulation must be repeated multiple times in order to ensure the statistical significance of the final results (e.g. in the experiments of figure 1 we repeat the simulation 100000 times). Thus, in the next section we address this practical issue by presenting an alternative algorithmic approach.

5. ALGORITHMIC COMPUTATION OF THE EXPECTED REVENUE

Given the comments above, we now present an algorithm that allows us to compute the revenue of our new bundle auction, in the case that the items are similar. This algorithm is exact, is much faster than the simulation approach presented in the previous section, and scales extremely well (i.e. as the number of bidders participating in the auction increases, the corresponding increase in computational cost is minimal). However, it is restricted to cases where the valuations of each item are i.i.d. and are drawn from the same distribution, $f(u)$. (for other cases the simulations are still the best method) This constraint implies that all of the items have similar values. While not always applicable, it certainly applies to our motivating example of selling a number of DVD movies when these movies are of the same class (e.g. new blockbusters).

To compute the expected revenue of the pick-a-bundle auction, we first need to calculate the probability density function that describes the bidders' valuation for the subset of k items that they value most highly (and hence will bid for). Since the top k order statistics are not independent, this is not at all trivial, and in fact, algorithms that can compute this pdf accurately for general valuation distributions and arbitrary numbers of items do not exist.[11] Here, we propose an algorithm that is polynomial with respect to variables k, m and L; it is shown in pseudo-code in figure 2. Once this pdf is computed, we compute the expected revenue of the pick-a-bundle auctions in time polynomial to n.

The main idea behind our algorithm is that we use three "for" loops to execute the computation. The main loop changes the value of variable x, which represents the value of the k^{th} order statistic. Once this is known, we use the other two loops to enumerate all the possible values of variables ω_b and ω_a, which represent the number of order statistics, which are respectively below (i.e. $(k+1)^{th}$ to m^{th} order statistics) and above[12] (i.e. 1^{st} to k^{th} order statistics)

[9]We selected to present here an example with many different distributions and valuations that are not always independent of each other (here u_{1j} and u_{2j} are not), in order to show how generally applicable our work is. In all the cases that we considered, the same broad pattern of results were observed.

[10]Based on the closing price of these auctions, the proper listing cost would be $C = 0.35$, if these were eBay auctions. The listing fees saved through bundling are equal to $(\lambda - 1) \cdot C$, where $\lambda = k$ for the pick-a-bundle auction, $\lambda = 1$ for separate auctions, and $\lambda = m$ for the complete bundle.

[11]There are asymptotic methods that give approximations of this pdf as the size of the problem increases to infinity [15]. However, we consider relatively small numbers of items, and these methods do not provide sufficiently accurate approximations in these cases.

[12]Actually here we also count the k^{th} order statistic as well, so it is $\omega_a \geq 1$. When we say that some order statistic is above the k^{th}, this means it is one of the 1^{st} to $(k-1)^{th}$ order statistics and

1. $p_a := 1, p_b := 0, p_{eq} := 0, g_m^k(u) = 0, \forall u$
2. for $x := 0$ to L do
3. $\quad p_b := p_b + p_{eq}$ [Note: $p_b = \sum_{i=0}^{x-1} f(i)$]
4. $\quad p_{eq} := f(x)$
5. $\quad p_a := p_a - p_{eq}$ [Note: $p_a = \sum_{i=x+1}^{L} f(i)$]
6. $\quad \vec{h} := [f(x+1) \ \ldots \ f(L)]$
7. $\quad \vec{h}' := [1]$
8. $\quad$ for $\omega_a := k$ to 1 step -1 do
9. $\quad\quad p := 0$
10. $\quad\quad$ for $\omega_b := 0$ to $(m-k)$ do
11. $\quad\quad\quad p := p + \frac{m!}{(m-k-\omega_b)!(\omega_a+\omega_b)!(k-\omega_a)!} p_{eq}^{\omega_a+\omega_b} p_b^{m-k-\omega_b}$
12. $\quad\quad$ end for loop (variable ω_b)
13. $\quad\quad$ if $(\omega_a = k)$ then increase $g_m^k(k \cdot x)$ by p
14. $\quad\quad$ else
15. $\quad\quad\quad \vec{h}' := \vec{h}' \otimes \vec{h}$
16. $\quad\quad\quad$ Increase $g_m^k(i + k \cdot x + k - \omega_a)$ by $p \cdot \vec{g}'(i), \forall i$
17. $\quad\quad$ end else
18. $\quad$ end for loop (variable ω_a)
19. end for loop (variable x)

Figure 2: Algorithm for computing the probability distribution of the sum of the k top order statistics of m i.i.d. variables drawn from distribution $f(u)$

the k^{th} order statistic x, and that have a value equal to x. The probability that $(\omega_a+\omega_b)$ variables have value equal to x, $(k-\omega_a)$ have values greater than x and $(m-k-\omega_b)$ have values smaller than x, is:

$$p(x,\omega_a,\omega_b) = \frac{m!(\sum_{i=0}^{x-1} f(i))^{m-k-\omega_b} f(x)^{\omega_a+\omega_b} (\sum_{i=x+1}^{L} f(i))^{k-\omega_a}}{(m-k-\omega_b)!(\omega_a+\omega_b)!(k-\omega_a)!} \quad (1)$$

The precise values that the bottom $(m-k-\omega_b)$ order statistics have are not important, because they correspond to the least valued items; only the values of the top $(k-\omega_a)$ order statistics must be taken into account. What we know is that in this case, all these values are greater than x. Therefore, we can take the pdf $h(u)$ to denote the conditional probability when we know that $u > x$. This is given by:

$$h(u) = \frac{f(u)}{\sum_{i=x+1}^{L} f(i)}, \text{ if } u \in \{x+1, \ldots, L\} \quad (2)$$

and $h(u) = 0$, otherwise. Given that the valuations are i.i.d. we can compute the distribution of their sum by taking the convolution:

$$h_{k-\omega_a}(u) = \underbrace{h(u) \otimes \ldots \otimes h(u)}_{(k-\omega_a) \text{ times}} \quad (3)$$

Now, we also know that another ω_a order statistics are equal to x, meaning that the pdf of the k top order statistics for this case is:

$$\widetilde{h}_{k-\omega_a}(u + \omega_a x) = h_{k-\omega_a}(u), \forall u \in \{0, \ldots, (k-\omega_a)L\} \quad (4)$$

and $\widetilde{h}_{k-\omega_a}(u) = 0$, otherwise. As this case happens with probability $p(x,\omega_a,\omega_b)$, and we account for all possible values of x, ω_a and ω_b with the three loops, the pdf of the sum of the top k order statistics is finally given by:

$$g_m^k(u) = \sum_{x=0}^{L} \sum_{\omega_a=1}^{k} \sum_{\omega_b=0}^{m-k} p(x,\omega_a,\omega_b) \widetilde{h}_{k-\omega_a}(u) \quad (5)$$

It should be noted that the algorithm in figure 2 has some additional optimizations (which are not described here due to limited space). Its complexity depends on k, m and L and is $O(k^2 m L^3)$, because we use a convolution algorithm with quadratic cost $O(kL^2)$. An even more efficient algorithm could make use of a Fast Fourier Transform to compute the convolution, which would reduce the complexity to $O(k^2 m L^2 \ln(kL))$.

Given that the probability distribution of the sum has been computed, the expected revenue, ER, of each auction is given by:

$$ER_G = n(n-1) \int_0^{mL} [1 - G(u)]^{n-2} g(u) u G(u) du \quad (6)$$

where $G()$ and $g()$ are, respectively, the cdf and pdf of the distribution of the valuation of the commodity sold in the auction; hence $g = g_m^k$ for the pick-a-bundle auction, and $g = f$ for the separate auctions. This can be done in $O(mL\ln(n))$ time, since the computation of a power x^n is done in $O(\ln(n))$ time. Thus, the complexity of computing the revenue of one pick-a-bundle auction is $O(k^2 m L^2 \ln(kL) + mL\ln(n))$, which is low polynomial complexity with the size of the problem. Consequently, this allows us to compute the best auction very fast and efficiently.

6. ACCOUNTING FOR THE SECOND ROUND OF AUCTIONS

In this section, we examine the effect of the second round of auctions on the buyers' bids, which has not been included in the work presented until this point. Initially, we examine dynamic settings, where the effect of the second round can be ignored (thus for these scenarios the analysis of the previous sections can be applied as presented), and then we examine settings in which it cannot. For the second case, we first compute accurately the bid shading that occurs due to the second round of auctions in the case where similar items are being sold (as discussed in the previous section) and we then proceed to examine more general settings.

In more detail, online auctions constitute *dynamic scenarios*, in which there are many external opportunities for buyers to acquire equivalent items (e.g. from other sellers in an online auction house such as eBay), and thus, the second round of the pick-a-bundle auction does not represent the last chance to acquire any particular item. In this case, there is no bid shading since its effect is already incorporated within the buyers' valuations for the items:

Claim 1 *In a dynamic auctions setting, we can ignore the effect of bid shading due to the second round of auctions.*

DISCUSSION. We can assume that the true internal valuations of bidder i in this setting are $\overline{u}_{i,j}$, which are drawn from a distribution with pdf $\overline{f}_j(u)$ for each item j. Since online auctions are often dynamic scenarios where bidders and buyers come and go and the same item is being sold in many different auctions, the bidders will indeed shade their bids in any auction in which they participate, because of the possibility of getting the item that they are interested in from another seller (in another auction). To prove this fact more rigourously, we will use the sequential auction model and the subsequent analysis presented in [14]. This analysis provides the mapping $\overline{g}_j : \overline{u}_{i,j} \rightarrow u_{i,j}$ from the real valuations $\overline{u}_{i,j}$ to the "shaded" ones $u_{i,j}$. Using the same mapping, we can generate the new "shaded" prior distributions $f_j(u)$ from which the shaded valuations $u_{i,j}$ are drawn. Because the number of potential sellers is sufficiently large in an online auction setting, following the analysis in [14], we can conclude that the bidder's bids will not change (more than a tiny amount), because of the presence of one more seller, or, equivalently, one additional chance to buy the item. Therefore, we can use the shaded valuations $u_{i,j}$, in our analysis of how bidders will bid in the pick-a-bundle format, to represent the valuations of bidders in both rounds of bids; the effect of additional bid shading is minuscule, if not non-existent. □

therefore its value is the same or higher than that of the k^{th} order statistic.

However, there are also *static scenarios*, in which a fixed number of buyers are participating and the items being sold are not available from any other sources (i.e. because they are rare or difficult to find). Within our pick-a-bundle auction, the buyers will thus have two opportunities to acquire each item: in the pick-a-bundle auction as part of a bundle of k items, and in the second round of separate single-item auctions, if that item was not sold in the bundle auction. Because of this second opportunity to purchase items if they don't win, buyers shade (i.e. strategically reduce) the bids placed in the bundle auction. We can initially consider the case of similar items:

Proposition 3 *Assume w.l.o.g. that $u_{i1} > u_{i2} > \ldots > u_{im}$ are the valuations of buyer i for the items offered for sale. If he participates in a pick-a-bundle auction where k items are sold in the first round, then, because of the chance to get the remaining $(m-k)$ of these items in the second round of auctions, he will bid:*

$$b_i = \sum_{j=1}^{k} \Big(u_{ij} - \frac{m-k}{m} \sum_{x=0}^{u_{ij}-1} \Omega(x) \Big) \quad (7)$$

where $\Omega(x)$ is the distribution of the top bid of the opponents it will face in the second round.

PROOF. Since the seller sells k out of m items in the pick-a-bundle auction initially (first round), then there is a $\frac{m-k}{m}$ chance that any item that a buyer wishes to purchase might be available for sale in the second round (i.e. there is an equal probability that each item is not among the top k most desired items of the winner of the first auction and therefore left for sale in the second round). Let us assume that item j for which a certain buyer i has valuation u_{ij}, is relisted in the second round. From his point of view, buyer i competes against a number of other bidders, whose maximum bid is B (the top order statistic of these bidder's private valuations). Since we assumed that B is drawn from distribution with pdf $\Omega()$, it is $Prob[B \leq x] = \Omega(x)$. Then bidder i with valuation u_{ij}, will bid truthfully and win this auction with probability $\Omega(u_{ij})$. The payment he will make is equal to B. Thus, his expected profit in this auction is given by:

$$EProfit_{ij} = \sum_{x=0}^{u_{ij}-1} (u_{ij} - x) \cdot Prob[B = x] = \sum_{x=0}^{u_{ij}-1} \Omega(x) \quad (8)$$

Therefore, if we assume w.l.o.g. that $u_{i1} > u_{i2} > \ldots > u_{im}$ are the valuations of buyer i, then he has a chance to win each of his k most valued items in the second round, if he does not win in the first. His expected total profit in this case is:

$$C_i = \frac{m-k}{m} \cdot \sum_{j=1}^{k} EProfit_{ij} = \frac{m-k}{m} \cdot \sum_{j=1}^{k} \sum_{x=0}^{u_{ij}-1} \Omega(x) \quad (9)$$

Now this buyer i will gain a profit of $(\sum_{j=1}^{k} u_{ij} - t)$, where t is the payment and is equal to the second highest bid, if he wins the pick-a-bundle auction, and an expected profit C_i if he loses it (since he then has a chance to buy some of the items he desires in the second round auctions). It is then trivial to show that the bid that maximizes his overall expected profit is $(\sum_{j=1}^{k} u_{ij} - C_i)$, and thus, he should shade his true valuation for the bundle (given by $\sum_{j=1}^{k} u_{ij}$), by an amount equal to his expected profit C_i. □

This effectively means that each bidder shades his true valuation u_{ij} for each item by $\frac{m-k}{m} \sum_{x=0}^{u_{ij}-1} \Omega(x)$.[13]

[13]In the case that this bidder faces $\mathcal{N} = n-1$ other bidders, the valuation of each being drawn from distribution with cdf $F(u)$, then the top opponent bid is drawn from distribution with cdf $\Omega(x) = F(x)^{n-1}$. This is not exactly the distribution that we will use in our experiments in section 7, because we need to account for the fact that the valuations of one bidder (the winner of the pick-a-bundle auction) are drawn from a different distribution, which we denote $H(u)$. In this case $\Omega(x) = F(x)^{n-2}H(x)$. In the experiments presented in the next section, we used a simulation to compute $H(u)$. The expected revenue of the separate auctions selling the remaining items is now computed by taking the expected value of distribution $\Psi(x) = (F(x))^{n-1} + (N-1) \cdot H(x) \cdot (1 - F(x)) \cdot F(x)^{n-2}$, rather than equation 6.

Now, we describe how to modify the algorithm of figure 2 in order to incorporate the bid shading. The easiest way to do so is to modify the prior distribution of the valuations $f(u)$, and to map every value $u \in \{0, \ldots, L\}$ to $u - \frac{m-k}{m} \cdot \sum_{x=0}^{u-1} \Omega(x)$. Note, however, that this equation does not always give integer values. For example, value $u = 3$ might need to be mapped to 2.4. If the new pdf $\hat{f}(u)$, that we input in the algorithm, were generated from these mappings by rounding each value to the closest integer (i.e. 2.4 becomes 2), then the sum of bids could have a significant error (as much as $\lfloor \frac{k}{2} \rfloor$). To alleviate this, we can simply create k virtual valuations. For example, if f is defined at $\{0, 1, \ldots\}$ we can decide to define it at values $\{0, \frac{1}{k}, \frac{2}{k}, \frac{3}{k}, \ldots\}$ and then round the shaded bids to these values rather than the integer values. By doing this, we guarantee that the total error in the computation is less than $\frac{1}{2}$. As the bidders must bid integer values, this means that we would have to round the final bid b_i anyway, and therefore with this trick we can guarantee that our algorithm computes as a value either $\lfloor b_i \rfloor$ or $\lceil b_i \rceil$. The algorithm now should have complexity $O(k^5 m L^3)$, but by taking advantage of the sparsity of the pdf in this case, we derive an algorithm with lower complexity $O(k^3 \cdot m \cdot L^3)$.

The previous case described accounts for bid shading in the case that the items are similar. So it's now time to examine the case that they are not and therefore they have different priors $f_j(u)$. In this case, we assume that the winner of the first round (i.e. the pick-a-bundle auction) will always select his top valued items. Then we can extend the results of the previous subsection to get:

$$b_i = \sum_{j=1}^{k} \Big(u_{ij} - \pi_j \sum_{x=0}^{u_{ij}-1} \Omega_j(x) \Big) \quad (10)$$

where $\Omega_j(x)$ is now the distribution of the top opponent bid for the j^{th} item, which is computed in the same way as in the previous case, and π_j is the probability that this item will be available in the second round; this probability is computed from the distributions $f_j()$. Due to the assumption made that the k highest valued items are always selected, this is really a fair, albeit not absolutely accurate, estimate of the bid placed, unlike equation 7 which is accurate for the previous case. We would like to explain why, in a few cases, the winner will not select his k most valued items in the first round. Let us give an example of when this happens. Assume that the k^{th} top valued items of winning bidder i has value $u_{i,j_k} = 10$ and the prior for that item is $f_{j_k} = U[0, 10]$ (uniform). Also the next highest valued item (i.e. the $(k+1)^{th}$ order statistic among bidder i's valuations) has value $u_{i,j_{k+1}} = 9$ and the prior distribution for that item is $f_{j_k} = U[9, 12]$ (also a uniform distribution). In most cases, it would be in bidder i's interest to select item j_{k+1} rather than j_k to get from the pick-a-bundle auction, even though its value is lower. This is due to the fact that this bidder will have a much easier time winning item j_k compared to item j_{k+1} in the second round, and thus make even more profit.

Even though this case of a winner not selecting one of his k highest valued items in the first round is not very common, it does complicate the analysis quite a bit. This is the reason why, we leave this analysis for future work.

7. EMPIRICAL RESULTS

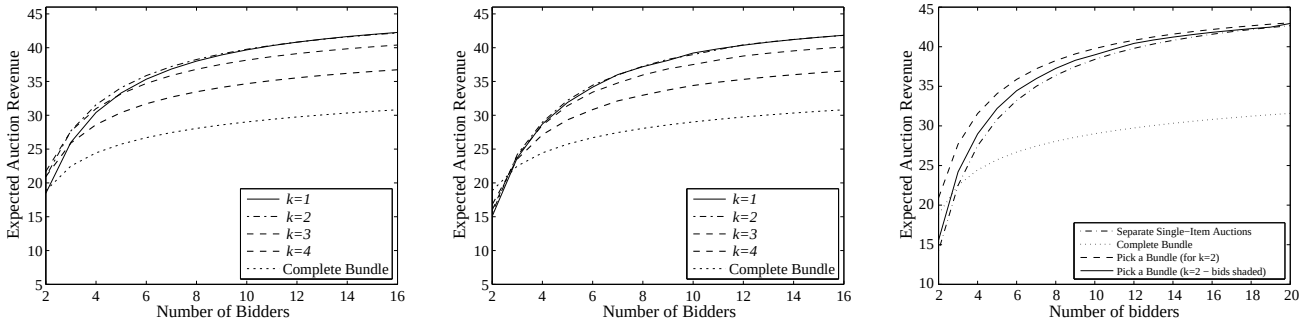


Figure 3: Seller's revenue for a bundle of $m = 5$ items, as the number of participating buyers n increases. In (b) the bids are shaded, whereas in (a) they are not. In (c) we include only the best pick-a-bundle auction ($k = 2$). The distribution of the buyer valuations $f(u)$ is uniform for values $\{0, \ldots, 9\}$.

In section 4, we showed how to compute the best bundle size for the pick-a-bundle auction and determine whether this is the best choice available to the seller. In this section, we give additional empirical results that make use of both the algorithmic computation (from section 5), which is faster and more efficient, and our analysis of the strategic buyer behaviour (from section 6).

The first experiment uses the algorithms of figure 2 and its extension which accounts for the buyers shading their bids for several scenarios (i.e. distributions) when all the items are similar. More specifically, we examine the seller's revenue, who has $m = 5$ items for sale, to n participating bidders. The valuations of the items are i.i.d. variables drawn from the same distribution $f(u)$. In figure 3 we graph the expected revenue, when $f(u)$ is a uniform distribution for values $\{0, \ldots, 9\}$, so $f(u) = 0.1, \forall u \in \{0, \ldots, 9\}$, and $f(u) = 0, \forall u \notin \{0, \ldots, 9\}$. We use uniform distributions, because they are the ones most commonly used in analyzing auction scenarios in the literature. We have also done the same analysis with several other distributions[14] and, though we don't present the results here, they lead to similar conclusions as those drawn here.

In figure 3(a), the expected revenue of the pick-a-bundle format is examined for all the values of the bundle size k,[15] when no bid shading occurs. The expected revenue of all cases is computed using equation 6. This figure allows us to consider how the choice of the bundle size (i.e. the value of k) affects the revenue of the auction, and hence, show how to determine which bundle size is optimal in any given case. As seen in the plot, the pick-a-bundle auction with $k = 2$ generates more revenue over a wide range of the number of bidders. More specifically, it yields the highest revenue for $n \in \{3, \ldots, 19\}$ and is thus the best choice for these values; when $n = 2$ (resp. $n \geq 20$), then the best choice is $k = 3$ (resp. $k = 1$). These results must also be set against the $(k - 1) \cdot C$ listing fees saved through bundling. In this case, $k = 2$ is the best choice when $n \geq 4$, while $k = 3$ is better for very small numbers of participating bidders, such as $n \leq 3$.

Figure 3(b) presents the equivalent results to figure 3(a), when bid shading does take place. The results in this case are very similar to those of figure 3(a); the pick-a-bundle auctions for $k = 1$ and $k = 2$ give almost the same expected revenue for most values of n (although, of course in practice, $k = 2$ would be preferred due to the listing fee savings). For very small values of n ($n \leq 3$), $k = 3$ can be the best choice, especially if listing costs are considered.

Based on the results of figures 3(a) and 3(b), the overall best choice for the bundle size is $k = 2$. We use this value in figure 3(c), where we compare the revenue obtained from the pick-a-bundle auction with those of the complete bundle and of using separate single-item auctions. When bid shading does not occur, the new bundling strategy is the clear winner, and when bid shading is taken into account it still outperforms the other two in almost all cases (the exception being when $n = 2$, when one should select the complete bundle). If we consider listing costs these observations do not change; for small values of n the complete bundle might be better (and only in the case where the bids are shaded) and for every other case the pick-a-bundle format is the best choice.

The second experiment is the same as that presented in section 4. Here, however, we also include the effect of bid shading and, thus, the buyers' bids are computed using equation 10. The results are presented in figure 4 and they reinforce the observations made in the previous experiments. Furthermore, comparing with figure 1, we notice that, while accounting for the bid shading reduces the benefit of using pick-a-bundle compared to the complete bundle and the separate auctions, it is still the best choice. More specifically, $n = 2$ bidders is the only case in which the complete bundle would be preferable (especially when including listing costs), while for small number of bidders ($n = 3, \ldots, 6$) the pick-a-bundle with size $k = 2$ is best, and for $n > 6$ bidders the pick-a-bundle of size $k = 1$ yields that highest revenue (even accounting for the listing costs). In addition, when the number of bidders becomes significant ($n \geq 14$ in this experiments) the difference between the pick-a-bundle and the separate auctions becomes small (less than the standard error for the number of samples that we used in our simulations). From all these, we can conclude that pick-a-bundle is the best option, with the optimal bundle size depending on the number of participating bidders n; however when there is sufficient competition (a large number of bidders n) the best choice is to use pick-a-bundle with bundle size $k = 1$ closely followed by separate auctions.

General Observations In general, from all the experiments we conducted, both the representative ones we presented in this paper and others we also performed, we observe that the complete bundle

[14]For example, as there is evidence to suggest that in many cases, there is a substantial probability that a buyer is either not interested at all or is interested very little in a specific item, e.g. the experimental distribution used for the TAC game analysis in [16], we used (among others) distributions that place substantial probability at value 0. This is a situation which is very likely to occur within the motivating example concerning bundles of DVDs; for example, if the buyer already has a copy of a DVD, he is likely to have little interest in acquiring a second one.

[15]The reader is reminded that selecting $k = m$ is the complete bundle case.

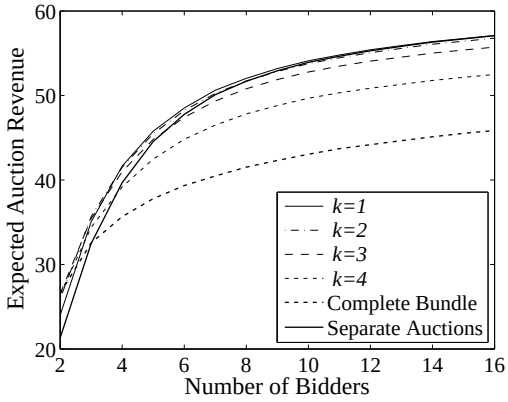


Figure 4: The same experiment as that of figure 1, when bid shading is also considered.

only makes sense in the case of few bidders ($n = 2$) and only in cases that the effect of listing costs and bid shading on the auction revenue is considered. The cases when relatively few (i.e. $n \leq 10$) bidders participate is when the pick-a-bundle gives the highest benefit compared, in particular, to the case of separate single-item auctions. While bid shading causes a drop of the pick-a-bundle revenue, this mainly happens when the number of bidders is small, because the chance of winning an item in the second round is highest for a small number of opponent bidders; and even with this effect, the pick-a-bundle is better (provided that the correct bundle size is choosen) than separate auctions. Finally, when we know that the number of bidders n is large (the exact number depending on the scenario), then, while the pick-a-bundle format with bundle size $k = 1$ is best in most cases, the revenue improvement over that of selling in separate auctions is small. The observations should provide a rule of thumb in the case that the seller would not wish to perform the complete analysis and yet still make a good choice as to the auction format that would yield maximal revenue.

8. CONCLUSIONS

In this paper, we examined the revenue of the auctioneer, when she sells items in bundles, rather than individually. Because we want these results to be applicable within existing online auctions, we rule out the design of new auction mechanisms; this is the main reason why the preexisting literature on bundling is not applicable here. On eBay, for example, it not uncommon to see a seller listing bundles of several similar items (i.e. DVDs), in order to save on the listing costs. However, our analysis has shown that this does not yield a higher profit for the seller due to the loss in expected revenue that is incurred. To address this loss, we propose the novel pick-a-bundle strategy, in which the seller lists the items in a single bundle, but the buyers bid for the opportunity to select a predetermined number of items from this bundle, with the remaining unsold items being sold in subsequent separate single-item auctions. We showed that not only does this policy reduce the loss in revenue incurred by the complete bundle, but it usually generates greater revenue than using separate single-item auctions as well (even before the savings in terms of listing fees are considered). This occurs because the pick-a-bundle auction induces additional competition between buyers who may actually prefer distinct subsets of the items. To compute the optimal bundle swiftly and accurately we developed the novel algorithm of figure 2. This algorithm can also be used in a variety of other settings, e.g. to compute the distributions of (i) the revenue that a seller makes in a position auction (Google Adwords is a variation of a position auction) from the distributions of the bids, and (ii) the social welfare (sum of winners' valuations) in a multi-unit auction. Furthermore, we examined how the strategic behaviour of the buyers changes the expected revenue of the pick-a-bundle auction and incorporated it into our analysis.

As future work, we will extend our analysis of the bid shading to the complete case of heterogeneous items. We also plan to examine what happens if we add an extra option to the bundle auction, for the winner of the first round to be able to buy additional items, e.g. by paying a predetermined percentage of the closing price, or paying the expected profit of the seller if she were to relist these items. This has the potential of increasing the seller's revenue, especially in the static case, where the bidders shade their bids in the first round; if there exists some chance that the winner selects more items in the first round, then the probability of having a second chance to buy each item decreases, therefore bidders will shade their bids less in the first round. Furthermore, the costs (listing and shipping) would be reduced. This extension also ties as well with another extension: examining how shipping fees would affect the revenue. Furthermore, we would like to investigate the application of previous results that show how the valuation distributions of participating bidders can be learned through observation of previous auctions [8]. This will allows us to apply our autonomous auction agent in setting in which these distributions are not known.

9. ACKNOWLEDGMENTS

We would like to thank Dr. Edith Elkind for comments on this work. This research was undertaken as part of the ALADDIN project and is jointly funded by a BAE Systems and EPSRC strategic partnership (EP/C548051/1).

10. REFERENCES

[1] W. Adams and J. Yellen. Commodity Bundling and the Burden of Monopoly. *Quarterly Journal of Economics*, 90(3):475–498, 1976.
[2] Y. Bakos and E. Brynjolfsson. Bundling information goods: Pricing, profits, and efficiency. *Management Sci.*, 45(12):1613–1630, 1999.
[3] Y. Bakos and E. Brynjolfsson. Bundling and competition on the internet. *Marketing Science*, 19(1):63–82, 2000.
[4] W. Conen and T. Sandholm. Preference elicitation in combinatorial auctions. In *ACM EC'01*, pages 256–259, New York, USA, 2001.
[5] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial auctions*. MIT Press, 2006.
[6] A. Giovannucci, J. Rodrıguez-Aguilar, and J. Cerquides. Multi-unit combinatorial reverse auctions with transformability relationships among goods. In *WINE 2005*, pages 858–867, Hong Kong, 2005.
[7] P. Jehiel, M. Meyer-ter Vehn, and B. Moldovanu. Mixed bundling auctions. *Journal of Economic Theory*, 134(1):494–512, 2007.
[8] A. X. Jiang and K. Leyton-Brown. Bidding agents for online auctions with hidden bids. *Machine Learning J.*, 67:117–143, 2007.
[9] J. O. Kephart, C. H. Brooks, and R. Das. Pricing information bundles in a dynamic environment. In *EC-01*, pages 180–190, Oct. 2001.
[10] V. Krishna. *Auction theory*. Academic Press, 2002.
[11] A. Likhodedov and T. Sandholm. Approximating revenue-maximizing combinatorial auctions. In *AAAI-05*, pages 267–274, Austin, Texas, USA, 2005.
[12] R. P. McAfee, J. McMillan, and M. D. Whinston. Multiproduct monopoly, commodity bundling, and correlation of values. *The Quarterly Journal of Economics*, 104(2):371–383, 1989.
[13] D. C. Parkes. iBundle: An efficient ascending price bundle auction. In *ACM EC'99*, pages 148–157, Denver, Colorado, USA, 1999.
[14] M. Said. Sequential auctions with random arrivals. *(working paper), Yale University*, 2008.
[15] R. J. Serfling. *Approximation Theorems of Mathematical Statistics*. New York, John Wiley, 1980.
[16] I. A. Vetsikas, N. R. Jennings, and B. Selman. Generating Bayes-Nash equilibria to design autonomous trading agents. In *IJCAI-07*, pages 1543–1550, Hyderabad, India, 2007.
[17] S. Wu, L. Hitt, P. Chen, and G. Anandalingam. Customized bundle pricing for information goods: A nonlinear mixed-integer programming approach. *Management Science*, 54(3):608–622, 2008.

A complete algorithm for DisCSP: Distributed Backtracking with Sessions (DBS)

Pierre Monier, Sylvain Piechowiak and René Mandiau
LAMIH UMR CNRS 8530
Université de Valenciennes et du Hainaut Cambrésis
F-59313 Valenciennes Cedex 9, France,
{pierre.monier;sylvain.piechowiak;rene.mandiau}@univ-valenciennes.fr

ABSTRACT

Many algorithms for Distributed Constraints Satisfaction Problem (DisCSP) resolution use additional links between variables not connected by constraints. This causes a higher needed memory space. In this paper, we propose an algorithm for DisCSP resolution, called Distributed Backtracking with Sessions (DBS) which does not use such additional links so that the initial problem's topology is respected. This algorithm is complete and requires a low space complexity. The main feature of this algorithm is to use the concept of sessions to provide a context for the exchanged messages.

Keywords

MAS, Distributed CSP, session, Backtracking

1. INTRODUCTION

Early researches on Constraints Satisfaction Problems (CSP) began in Artificial Intelligence in the 1970s. The CSP formalism addresses many problems in a simple and efficient way such as road traffic management [2].

However, some problems which use physically distributed data cannot be solved in a classical and centralized way. The causes can be multiple :

- The time to gather stored data on a single site can be prohibitive.
- Too much memory space is needed to store the whole problem.
- The data exchanges on many sites can bee limited for confidentiality or safety reasons.

Distributed CSPs (DisCSP) were proposed in order to solve naturally distributed problems. The problem to be solved is thus distributed on all agents. These agents interact in order to find a global solution based on each agent's local solutions. DisCSP are usually applied to solve distributed problems such as timetabling [7].

More formally, a DisCSP is a 4-tuple (X, D, C, A) where: X is a finite set of p variables: $\{x_1, x_2, ..., x_p\}$, D is a set of domains associated with these variables $D = \{Dom(x_1), Dom(x_2), ..., Dom(x_p)\}$, C is a finite set of m constraints: $\{c_1, c_2, ..., c_m\}$ and A is a finite set of n agents: $\{A_1, A_2, ..., A_n\}$ where each agent is given a subset of X.

Many algorithms for DisCSP resolution have been described. The first and most known is ABT [8]. It is often used as a reference in many papers such as [1].

In this paper, we propose a complete algorithm to solve variable-based distributed problems (different from AAS [6] which is constraint-based). Our algorithm, called Distributed Backtracking with Sessions (DBS) does not add links between agents during the search.

Unlike most DisCSP algorithms such as ABT, DBS priority does not consist in minimizing the number of exchanged messages but aims at minimizing the time required to process a message. The less an agent requires CPU time to process a message, the less the message will be exact and complete. This brings about an augmentation of the total number of messages (but less costly messages) to solve the DisCSP. The work exposed in this paper is based on this trade-off

Another characteristic of DBS consists in using the concept of sessions to provide a context for each exchanged message. Moreover, these sessions allow the use of heuristics to remove obsolete messages containied in the inboxes (messages awaiting process) of each agent without eliminating possible solutions from the problem. Consequently, the number of exchanged messages is reduced.

The remainder of this paper is organized as follows. Section 2 presents our contribution for the DBS algorithm initially described in [3, 5]. The properties of this algorithm are exposed in section 3. Section 4 describes different heuristics used to delete obsolete messages. Performances of DBS with and without the proposed additional heuristics are compared using a DisCSP generator. Finally, section 5 discusses conclusions and possible improvements of the proposed algorithm.

2. DBS

The algorithm called Distributed Backtracking with Sessions (DBS) is detailed in this section. We start by giving assumptions and notations required by this algorithm and we explain in detail how it works (section 2.1). The behavior of DBS is illustrated in section 2.2 using a simple example.

2.1 Algorithm

A total order, called $\succ$, is established between the different agents (a priority is assigned to each agent) to avoid infinite loop problems. For example, a change for A_1 implies

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

a change for A_2 which involves a change for A_1. Given two agents A_1 and A_2, $A_1 \succ A_2$ means that A_1 has a higher priority compared to A_2. In this paper, agent's priority is given according to the lexicography order ($A_1 \succ A_2 \succ \cdots \succ A_n$). Given a generic agent called $self$, $\Gamma^-(self)$ (respectively $\Gamma^+(self)$) represents $self$'s higher priority (respectively lower priority) acquaintances, those acquaintances being connected with $self$ by a constraint.

In this paper, only one variable is assigned to each agent. Each agent assigns (in a concurrent way) a value for its variable and sends it to agents in $\Gamma^+(self)$ with an "ok?" message. If an agent $self$ cannot find an instantiation in agreement with the received values (stored in a set called $agent\_view(self)$), $self$ informs an agent $A_i \in \Gamma^-(self)$ with a "backtrack" message.

To determine if a message is obsolete or not, the session concept is used: a session number is attached to each message.

DEFINITION 1. *Given an agent $self$ and $\Gamma^+(self)$ the set of acquaintances whose priority is lower than $self$'s priority. A session between $self$ and $\Gamma^+(self)$ is an integer indicating for each element of $\Gamma^+(self)$ the state of the global search from $self$'s point of view.*

During the initialization step of the algorithm, the session of each agent is set to 0. When $self$ receives a backtrack message, called m_b, this one is processed only if the session number attached to m_b is equal to $self$'s session number. Otherwise m_b is considered obsolete. When $self$ receives an ok? message, it closes its session (its session number is incremented).

For a given pair of agent, we suppose that messages are received in the same order they are sent. DBS uses three types of messages :

- ("ok?", (A_k, v_k, s_k)): This message contains a triple which is composed of the sender of this message (A_k), the agent's value (v_k) and the agent's session (s_k).
- ("nogood",(A_k, v_k, s_k), $listeBT$). Backtrack request addressed to the agent A_k for its value v_k in the session s_k. A set of triple (x_i, v_i, s) called $listeBT$ is attached to this request. $listeBT$ is used by A_k to proceed, if needed, a backtrack message to an agent contained in $listeBT$.
- ("stop"): No solution exists and the receiver agent stops.

The context of an agent, called $self$, is defined by :

- $Self$'s current value.
- $Self$'s current session.
- A set called *propose* used to know, for each value $v_i \in D_{self}$, if v_i has already been transmitted to $\Gamma^+(self)$ in the $self$'s current session.
- A set called $agent\_view$ containing triple (x_i, v_i, s_i) used to determine the value v_i and the session s_i received from agents $x_i \in \Gamma^-(self)$.
- An integer set called $receivedBtVal$ used to know, in the $self$'s current session, values in D_{self} which have already received a backtrack request.
- A set called $totalBtSet$ containing triple (x_i, v_i, s_i). When $self$ has to send a backtrack request, this set allows $self$ to transmit, if needed, a backtrack message addressed to an agent in $totalBtSet$.

There are two different ways to terminate DBS. If a solution exists, there are no more exchanged messages between agents. This stable state allows to affirm that a solution has been found. If no solution exists, an agent will detect this lack of solution and will send a stop message.

Given A and B two sets of triples (s, v, s), we note: $A \uplus B = A \cup \{(x, v, s) \in B | (x, \_, \_) \notin A\}$. This operator is used by DBS algorithm (for example, to update $agent\_view$).

DBS algorithm is composed of different procedures numbered from 1 to 6 and presented below. Procedures 1 and 2 are respectively used when $self$ receives an ok? or a nogood message.

Procedure 1 is used to update $agent\_view$ (line 1), to close $self$'s session (line 2) and to check if there exists a value which is consistent with $agent\_view$ (line 3).

Procedure 2 checks if a received nogood message is not obsolete (line 1). $receivedBtVal$ and $totalBtSet$ are updated (lines 2 and 3), then $self$ checks again if there exists a value which is consistent with $agent\_view$. When $self$ receives a stop message, it terminates.

Procedures 3 and 4 are respectively used to close $self$'s session and to submit $self$'s current assignation.

Procedure 5 is used to check if a value in D_{self} which is consistent with $agent\_view$ in order to transmit an ok? message (line 4) or a backtrack message (lines 2 and 6).

Procedure 6 is used when it is necessary to send a backtrack request. The code given from lines 1 to 10 is used to determine the information to be attached to the backtrack request. If DisCSP is inconsistent, the triple (x, v, s) (line 12) will be equal to $null$ for a given agent and a stop message will be transmit. After the emission of the backtrack message, $totalBtSet$ is updated (line 17). If the backtrack message has been sent to an agent in $agent\_view$, this set is updated (line 19), else $self$ closes its session and then submits an assignment to $\Gamma^+(self)$ (lines 21 and 22).

Algorithm 1 when received (**"ok?"**, (x_j, d_j, s_j)) from A_j **do**

```
1: agent_view ← {(x_j, d_j, s_j)} ⊎ agent_view
2: close_session()
3: check_agent_view(A_j,"ok?")
```

Algorithm 2 when received (**nogood**, (x_j, d_j, s_j), $BtSet$) **do**

```
1: if s_j = current_session ∧ d_j ∉ receivedBtVal then
2:    receivedBtVal ← receivedBtVal ∪ {d_j}
3:    totalBtSet ← BtSet ⊎ totalBtSet
4:    if d_j = current_value then
5:       current_value ← null
6:    end if
7:    check_agent_view(null,"backtrack")
8: end if
```

Algorithm 3 close_session

1: $current\_value \leftarrow null$
2: $current\_session \leftarrow current\_session + 1$
3: $receivedBtVal \leftarrow \emptyset$
4: **for all** $d \in D$ **do**
5: $propose[d] \leftarrow false$
6: **end for**

Algorithm 4 submit_assign

1: select $d \in D \mid \neg propose[d] \wedge$ consistent$(d, agent_view)$
2: $current\_value \leftarrow d$
3: $propose[d] = true$
4: send(**"ok?"**, $(x_{self}, current\_value, current\_session)$) to outgoing links

Algorithm 5 check_agent_view(Agent: A_k, String: $type$)

1: **if** $A_k \neq null \wedge (\forall d \in D, propose[d] \vee \neg consistent(d, \{(A \succeq A_k, \_, \_) \in agent\_view\}))$ **then**
2: **backtrack**$(A_k, type)$
3: **else_if** $\exists d \in D \mid \neg propose[d] \wedge$ consistent$(d, agent_view)$ **then**
4: submit_assign()
5: **else**
6: **backtrack**$(null, type)$
7: **end if**

Algorithm 6 backtrack(Agent: A_k, String: $type$)

1: **if** $A_k \neq null$ **then**
2: $(x, v, s) \leftarrow \{(x', v', s') \in agent\_view | x' = A_k\}$
3: $BtSet \leftarrow \{(x', v', s') \in agent\_view \uplus totalBtSet | x' > x\}$
4: **else_if** $type =$ "$ok?$" **then**
5: $(x, v, s) \leftarrow \{(x', v', s') \in agent\_view | \forall A \in agent\_view, A \succeq x'\}$
6: $BtSet \leftarrow \{(x', v', s') \in agent\_view \uplus totalBtSet | x' > x\}$
7: **else**
8: $BtSet \leftarrow agent\_view \uplus totalBtSet$
9: $(x, v, s) \leftarrow \{(x', v', s') \in BtSet | \forall A \in BtSet, A \succeq x'\}$
10: $BtSet \leftarrow BtSet - \{(x, v, s)\}$
11: **end if**
12: **if** $(x, v, s) = null$ **then**
13: broadcast to other agents "stop" message
14: terminate this algorithm
15: **end if**
16: send (**backtrack**, $(x, v, s), BtSet$) to x
17: $TotalBtSet \leftarrow TotalBtSet - \{BtSet \cup (x, v, s)\}$
18: **if** $\{(x, \_, \_)\} \in agent\_view$ **then**
19: $agent\_view \leftarrow agent\_view - \{(x, v, s)\}$
20: **else_if** $type =$ "$backtrack$" **then**
21: close_session()
22: submit_assign()
23: **end if**

2.2 Example

A simple illustration of DBS algorithm is given in figure 1. This problem is composed of four variables $\{x_1, x_2, x_3, x_4\}$ connected by three constraints $(x_1 \neq x_3)$, $(x_1 \neq x_4)$ and $(x_2 \neq x_4)$. One possible execution of DBS appears in figure 2.

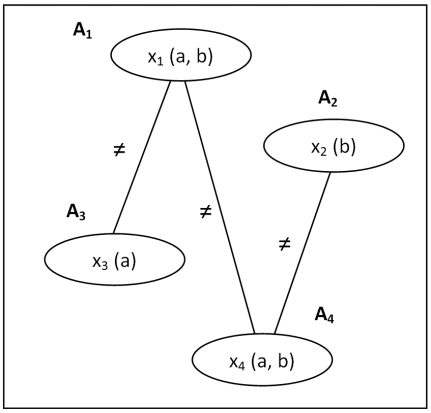


Figure 1: DisCSP example.

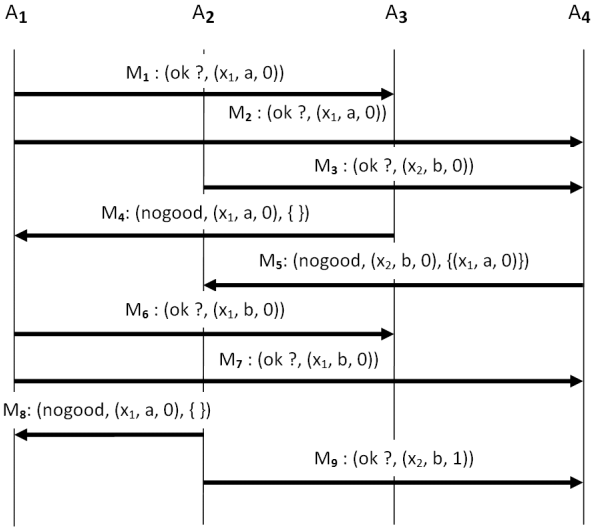


Figure 2: Echanged messages between A_i and A_j.

Initially, each agent chooses the first value of its domain D and then sends it to agents in $\Gamma^+(self)$. Indeed, M_1, M_2 and M_3 messages are sent. Upon receiving M_1, A_3 adds $(x_1, a, 0)$ to $agent\_view(A_3)$. Since A_3 cannot satisfy the constraint $(x_1 \neq x_3)$, it transmits a backtrack message (M_4) to A_1 then removes the triple containing A_1 from $agent\_view(A_3)$.

A_4 receives M_2 and M_3. Since it cannot find a possible instantiation in agreement with $agent\_view(A_4) = \{(x_1, a, 0), (x_2, b, 0)\}$, it sends a backtrack request to the lowest priority agent contained in $agent\_view(A_4)$, i.e. A_2. It attaches to this message (M_5) $BtSet = \{(x_1, a, 0)\}$ in order to allow A_2 to proceed, if needed, the backtrack request. Then A_4 removes $(x_2, b, 0)$ from $agent\_view(A_4)$.

A_1 receives M_4 and modifies its current value then sends its new value to A_3 and A_4 using M_6 and M_7 messages. A_2 receives M_5 and updates $totalBtSet$ with $BtSet(M_5)$. Since it cannot find a value, it transmits a backtrack request (M_8) to A_1 which is the lowest priority agent contained in

$agent\_view(A_2) \uplus totalBtSet$. A_2 forgets all previous references to A_1 because A_2 removes them from $totalBtSet$. Then, since A_2 has just sent a backtrack request to an agent not included in $agent\_view(A_2)$, it closes its session (the session become 1) and sends again its current value to agents in Γ^+ (here A_4) using M_9 message.

A_1 receives M_8 but does not process it because A_1 has already received a backtrack request about the same value in its current session (0). A_4 receives M_9. There are no more exchanged messages. The solution is found: $\{(x_1 = b), (x_2 = b), (x_3 = a), (x_4 = a)\}$.

3. DBS PROPERTIES

In this section, we will demonstrate that DBS is sound (section 3.1), complete and that it terminates (section 3.2). Then, we show that its spatial complexity is polynomial bounded (section 3.3). In order to prove the DBS's completeness, we use the completeness proof of both ABT [8] and DDB [1].

3.1 Soundness

THEOREM 1. *DBS is sound, in that it only claims a solution if one exists.*

PROOF 1. *Whenever DBS detects a solution, all agents are in a stable state, waiting for messages. Such a state is incompatible with constraint violations, which would entail at least one message.*

3.2 Completeness

To demonstrate that DBS is complete and terminates, we consider DBS_{all}, an alternate implementation of DBS (similar to ABT) with full Forbidden Instantiation Combination (FIC)[1] recording. The use of FIC is needed to prove the completeness. We will show that DBS_{all} is equivalent to ABT whose completeness has been proved [8]. Indeed, the DBS_{all} completeness proof will be shown. Then, in a second step, we will shown that it is not needed to store all FIC during the search. next, we will prove that DBS preserves all properties related to DBS_{all}. DBS completeness proof will be achieved.

3.2.1 DBS_{all} completeness proof

DBS algorithm uses temporarily stored Forbidden Instantiations (FI). A FI (for DBS) corresponds to a forbidden value by $self$'s variable. FI are stored in a set called $receivedBtVal$[2]. When $self$'s session changes, this set is re-initialized.

To prove DBS completeness, we will modify this algorithm so that it can record FIC during all the solution search. The objective aims to describe the same completeness proof than for ABT. DBS algorithm with FIC recording is called DBS_{all}. Now, we will explain why DBS_{all} is similar to ABT (in four steps).

Nogood (FIC) recording.

In ABT, when $self$ receives a *nogood*, it adds this nogood message to its set of *nogoods*. In DBS_{all}, when $self$ receives a backtrack request (**"backtrack"**, $(self, default\_value, session)$, $BtSet$), it checks $agent\_view$.

[1]A FIC for DBS_{all} is equivalent to a *nogood* for ABT.
[2]DBS_{all} stores FIC in a set called $FICset$.

Let us suppose $agent\_view(self) = \{(x_1, v_1, s_1), ..., (x_k, v_k, s_k)\}$. Unlike DBS which adds $default\_value$ to the set called $receivedBtVal$, DBS_{all} completes $default\_value$ in order to create a FIC. This FIC, which is stored in a set called $FICset$, is $\{(x_1, v_1, s_1), ..., (x_k, v_k, s_k), (x_{self}, default\_value, session)\} \cup BtSet$.

$FICset$ is never re-initialized. Now, DBS_{all} records FIC as ABT records *nogood*. Note that $FICset$ contains triple $(variable, value, session)$ but that would be exactly the same if it contained $(variable, value)$ pairs.

Backtrack message context.

Unlike ABT which attached a nogood set for each backtrack message, DBS_{all} uses the concept of session. The set called $BtSet$ attached to these messages is only used to proceed, if needed, the backtrack request.

Concerning with the behavior of ABT, suppose that $self$ receives the *nogood*: $\{(x_1, v_1), ..., (x_k, v_k), (x_{self}, default\_value)\}$. There will be a backtrack, for ABT, if the two following conditions are respected:

1. $default\_value$ is equal to $self$'s current value.
2. for each (x_i, v_i) pair in *nogood*, if x_i belongs to $agent\_view$ then v_i (of the *nogood*) must be equal to the value contained in $self$'s $agent\_view$.

Concerning with the behavior of DBS_{all}, suppose that $self$ receives (backtrack, $(self, default\_value, session)$, $BtSet$). There will be a backtrack, for DBS_{all}, if the two following conditions are respected :

1. $default\_value$ is equal to $self$'s current value.
2. session attached to this message is equal to $self$'s session.

For DBS_{all}, the first condition is the same as the first condition for ABT. The second one (for DBS_{all}) is true if $self$ does not receives an ok? message between the time it sends its current value and the time it receives a backtrack message on this same value (so called $default\_value$). In this case, $self$'s $agent\_view$ is not modified (it is equivalent to the second condition for ABT). Indeed, DBS_{all} uses a message context equivalent to the one used in ABT.

Choice of the backtrack message receiver (after receiving an ok? message).

ABT and DBS_{all} differ on the choice of the backtrack message receiver. For these two algorithms, when $self$ receives an ok? message from an agent A_k, $self$ update $self$'s $agent\_view$. Suppose that no value in $self$'s domain is compatible with $agent\_view$.

1. In ABT, $self$ build a *nogood* set with $self$'s $agent\_view$ subsets where no solution exists. Then, for each subset, a backtrack message is sent to lowest priority agents in the subset.
2. In DBS, if no partial solution is found (a value for $self$ which is consistent with $A_i \in agent\_view | A_i \succeq A_k$), the backtrack message is sent to A_k (line 2 of *backtrack* procedure). If a partial solution exists, the backtrack message is addressed to the lowest priority agent in $self$'s $agent\_view$ (line 5 of *backtrack* procedure).

ABT directly sends the backtrack message to the faulty agent; while DBS transmits the backtrack message to the faulty agent or to an agent having a lower priority than the faulty agent. It results in the exploration of irrelevant branches of the search tree but no solution is eliminated.

Backtrack message to $A_k \notin$ agent_view.

For both ABT and DBS algorithms, each agent knows the address of the other agents. Indeed, a message emitting is possible for all agent pairs.

In DBS_{all}, a backtrack message addressed to $A_k \notin agent\_view$ can appear when $self$ has to send a backtrack request upon the reception of a backtrack message. We know that each agent in DBS_{all} knows all other. Indeed, in a first step, we need to know the receiver of this backtrack message. Next, we need to define the message's content.

In ABT, the choice of the backtrack message receiver upon the reception of a backtrack message is based on the following method. When $self$ receives a *nogood* containing x_j ($x_j \notin \Gamma^-(self)$), $self$ adds x_j to $agent\_view$ and searches a solution. If there are no solution, $self$ searches inconsistent subsets from $agent\_view$, then, for each subset, $self$ transmits a backtrack message for the lowest priority agent and removes it from $agent\_view$ (this agent can be x_j).

In DBS_{all}, when $self$ receives a backtrack message m_b where $BtSet(m_b)$ contains an agent $x_j \notin self$'s $agent\_view$, $self$ does not add x_j to $self$'s $agent\_view$ opposite to ABT ($self$ adds x_j to $totalBtSet$). Not adding x_j to $agent\_view$ is similar to ABT behavior because even if $self$ adds it to $agent\_view$, since there are no constraint between $self$ and x_j, the search for a solution will be the same. Then $self$ searches a solution. If no solution exists, $self$ sends the backtrack message to the lowest priority agent in $agent\_view \uplus totalBtSet$. This is similar to the behavior of ABT.

Finally, we need to determine the content of the backtrack message. Suppose that the message must be transmit to x_j. In ABT, the value v_j from x_j (which is included in $self$'s $agent\_view$) is included to the *nogood* ready to be send. With DBS_{all}, the triple (x_j, v_j, s_j)[3] contained in $self$'s $agent\_view \uplus totalBtSet$ is included to the backtrack message. Moreover, in DBS_{all}, $self$ adds a set of triple (x, v, s) called $listeBT$ to the message so that x_j can proceed the backtrack message.

To conclude on the first step of the completeness proof, we have described an alternative implementation for DBS, called DBS_{all}, with full FIC recording during the search. We have shown that DBS_{all} is similar to ABT: nogoods recording (FIC for DBS_{all}), context of backtrack message, choice of the a backtrack message receiver following an ok? message and a backtrack message. ABT is complete and terminates [8] indeed is DBS_{all}. In section 3.2.2, we will prove that DBS algorithm (with removal obsolete forbidden instantiations) is complete too.

3.2.2 *DBS completeness proof*

DBS differs from DBS_{all} on FIC recording. In this section, we will prove that eliminating Forbidden Instantiations (FI) cannot cause an infinite loop between agents. This equivalent to proof that DBS terminates.

[3]We add the session s_j because DBS uses the concept of session to determine message context.

LEMMA 1. *Let A_1 be the agent with the highest priority. A_1 can never fall into an infinite loop because of the way obsolete Forbidden Instantiations (FI) are discarded.*

PROOF 1. *DBS re-initializes the set called $receivedBtVal$ containing FI when an agent $A_i \in \Gamma^-(self)$ sends an ok? message to $self$ or when $self$ sends a backtrack message to an agent $A_j \in \Gamma^-(self) \notin self$'s $agent\_view$. A_1 is the highest priority agent so $\Gamma^-_{A_1} = \emptyset$. Values stored in $receivedBtVal$ for A_1 can never be removed. Moreover A_1 have a finite domain, so the size of the set called $receivedBtVal$ will be, in the worst case, equal to d where d is the size of the A_1's domain. A_1 can never re-initialize $receivedBtVal$ and A_1 can only receive a finite number of values d, A_1 cannot fall into an infinite loop.*

LEMMA 2. *If the first $(k-1)$ agents, in the order defined by DBS, are not trapped in an infinite loop, A_k cannot fall into an infinite loop because of the way $receivedBtVal$ (containing obsolete Forbidden Instantiations FI) is reinitialized.*

PROOF 2. *Let us suppose A_k is actually looping. That means that it forgets FI because A_k's predecessors which continuously change their values. FI are removed from A_k (procedure **close_session**) when :*

- *A_k receives an ok? message from an agent in $\Gamma^-(A_k)$. A_k's session is so incremented.*
- *A_k sends a backtrack message to an agent A_i in $\Gamma^-(A_k)$ not included in A_k's $agent\_view$ because A_i has modified its value. A_k's session is so incremented.*

But since we assume that no agent among $A_1, ..., A_{k-1}$ is in an infinite loop, they will stabilize in a finite time. Indeed, A_k exits its so-called infinite loop or find there are no solution to the DisCSP. A_k is not in an infinite loop.

THEOREM 2. *DBS is sound, complete, and terminates.*

PROOF 2. *By recurrence about lemmas 1 and 2, we affirm that none of DBS agents can fall into an infinite loop, despite the fact that DBS discards obsolete Forbidden Instantiations (FI). DBS has the same properties than DBS_{all} and terminates.*

We have shown an alternative DBS implementation called DBS_{all}. This one is equivalent to ABT whose the completeness proof is known. We have proved that DBS_{all} is complete too. Then, we have shown that none of DBS_{all} agents can fall into an infinite loop. We have demonstrated that DBS is sound, complete, and terminates.

3.3 Spatial complexity

THEOREM 3. *Each agent performing DBS requires a polynomially bounded storage space.*

PROOF 3. *Section 2.1 underlines that each agent has a domain (of size d), a set called $agent\_view$ (of size n in the worst case), a set called $propose$ (of size d in the worst case), a set called $receivedBtVal$ (of size d in the worst case) and a set called $totalBtSet$ (of size n in the worst case). So, memory space needed for each agent is $O(d+n)$.*

Memory space needed for each agent performing DBS is ($O(d+n)$). It is better than both DDB ($O(d*n)$) and ABT ($O(d^n)$).

4. EXPERIMENTAL RESULTS

We describe four heuristics for DBS algorithm. These heuristics (section 4.1) allow a reduction of the number of exchanged messages during the solution search without eliminating solutions. Results obtained by DBS, described in section 4.2, are compared to ABT, a classical algorithm used as a reference.

4.1 Heuristics

Given a generic agent $self$, $self$'s inbox containing unprocessed messages is called IB_{self}.

HEURISTIC 1. *If $self$ has, in IB_{self}, several ok? messages from an agent A_k, only the last ok? message sent by A_k is processed, others are removed. This heuristic, called h1, does not remove any solutions.*

PROOF 1. *A session, for a given agent, cannot be decremented. If an agent A_k sends several ok? messages to $self$, then the last ok? message, called m_{last}, have an attached session superior or equal to others ok? messages sent by A_k.*

Suppose that several ok? messages sent by A_k (received by $self$) are contained in IB_{self} . The last received ok? message is: m_{last} = ("ok?", (A_k, v_k, s_k)). Four cases are possible for others ("ok?", (A_k, v_i, s_i)) messages, received from A_k :

1. *($v_i = v_k$ and $s_i = s_k$): Impossible because an agent cannot submit multiple times the same value during a given session.*
2. *($v_i = v_k$ and $s_i < s_k$): The context of a message is defined by a session number. Messages containing an obsolete session number are not processed. In this case, if $self$ processes the ok? message containing s_k (m_{last}), then ok? messages containing s_i become obsolete and can be removed.*
3. *($v_i \neq v_k$ and $s_i = s_k$): Each agent uses values in its domain in an order defined in the initialisation step. This order does not change during the solution search. Indeed, value v_i from session s_i is an obsolete value compared to v_k (in the same session). h1 does not remove any solution even if DBS uses (after the session closing) a value ordonnancing heuristic like those cited in [4]. In fact, if A_k sends a new value v_k, it is because it has received a backtrack message for v_i. Ok? messages containing s_i are obsolete compared to m_{last} and are removed.*
4. *($v_i \neq v_k$ and $s_i < s_k$): Messages containing $s_i < s_k$ are deleted (see second case).*

HEURISTIC 2. *If $self$ has in IB_{self}, many backtrack messages and, at least, an ok? message then backtrack messages are removed from IB_{self}. This heuristic, called h2, does not remove any solutions.*

PROOF 2. *Suppose that $self$ first processes an ok? message, $self$'s session is incremented. Backtrack messages contained in IB_{self} become obsolete because they contain a value from an obsolete $self$'s session. All backtrack messages are removed because they are not processed by DBS (line 1 from procedure 2).*

HEURISTIC 3. *If $self$ has sent a backtrack message to $A_k \in$ $self$'s agent_view, then as long as $self$ has not received an ok? message from A_k, $self$ removes all backtrack messages from IB_{self}. This heuristic, called h3, does not remove any solutions.*

PROOF 3. *Suppose that $self$ has sent a backtrack message to A_k. $Self$ will receive, in a finite time, an ok? message, called m_k, from A_k. If $self$ waits for this message (then processes it) before processing backtrack messages already contained in IB_{self} then, using h2, $self$ removes all received backtrack messages.*

HEURISTIC 4. *If $self$ have many ok? messages in IB_{self}, then $self$ processes, in priority, those coming for higher priority agents. This heuristic, called h4, does not remove any solutions.*

PROOF 4. *Using this heuristic, no message is removed from IB_{self}. DBS is complete (section 3.2), so this algorithm works for all message reception orders (even with the order obtained with h4). However, hypothese from section 2.1 must be respected: "For a given pair of agent, we suppose that messages are received in the same order they are sent".*

4.2 Results

We evaluate the efficiency of DBS algorithm using JADE multi-agent platform. Our purpose here is to obtain a not-deterministic scheduling agent. In many papers, a discrete event simulator is used: each agent is activated one after another using cycles. One cycle consists in reading all incoming messages (during a cycle t) and sending messages (available to others agents' inboxes in cycle $t+1$). In real conditions, agents are not activated one after another. Moreover, this method would favour DBS when it uses message suppression heuristics. Indeed, the more messages there are in inboxes (which have not already been processed), the more our heuristics are efficient.

Unlike ABT, DBS's priority does not consist in minimizing the number of exchanged messages but aims at minimizing the time required to process a message. Table 1 shows that DBS, for a DisCSP example, uses more messages than ABT but requires less CPU time. Moreover, the maximal number of messages contained simultaneously in the different inboxes and waiting for process is lower for DBS using heuristics than for ABT. For ABT, when an agent sends a backtrack message to an agent $A_k \in \Gamma^-$, messages will be processed with a longer time than for DBS using heuristics.

We performed experimental evaluations on DisCSP created with a randomly DisCSP generator. This one requires four parameters: the number of agents/variables (N), the domain's size for each variable (D), the percentage of constraints between agents (P_1) and the percentage of forbidden tuples per constraint(P_2). The number of constraints is $\frac{n\times(n-1)}{2} \times P_1$. The number of forbidden tuples per constraint is $d \times d \times P_2$. A disCSP is represented as a tuple $< N, D, P_1, P_2 >$.

For example, the following problems have been generated: Problems with a constraint density equal to 40% $< 15, 10, 0.40, P_2 >$ and equal to 80% $< 15, 10, 0.80, P_2 >$. For each problem, percentage of forbidden tuples for each constraint ranges from 10% to 90%. Each dot in the plots presented in figures 3 and 4 corresponds to the average CPU time (over

Table 1: $< 20, 10, 0.20, 0.60 >$ **DisCSP: average on 200 experiments.**

Used algorithm	CPU times (in seconds)	Number of checked constraints (in millions)	Number of exchanged messages	Maximal number of present messages in inboxes awaiting processing	Standard deviation (in seconds)
ABT	29.3	2.7	19 650	1 340	104
DBS	26.7	2.8	236 076	1 554	74
DBS using heuristics	14.8	1.3	101 5105	9	60

Table 2: $< 15, 10, 0.50, 0.50 >$ **DisCSP: average on 200 experiments.**

Used algorithm	CPU times (in seconds)	Number of checked constraints (in millions)	Number of exchanged messages	Maximal number of present messages in inboxes awaiting processing	Standard deviation (in seconds)
ABT	428,2	24,1	109 700	15 471	751
DBS	20,5	2,6	170 137	18 823	13
DBS using heuristics	9,3	0,8	44 915	10	6

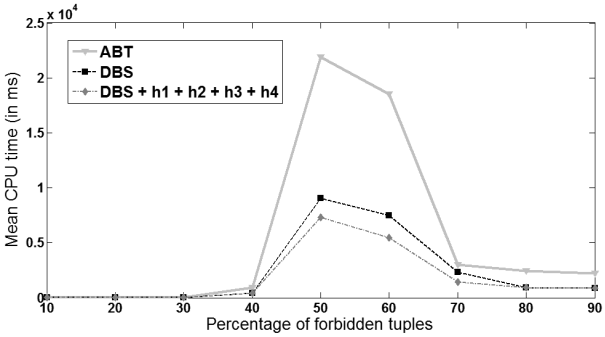


Figure 3: DisCSP constrained with 40% of the maximal constraint number.

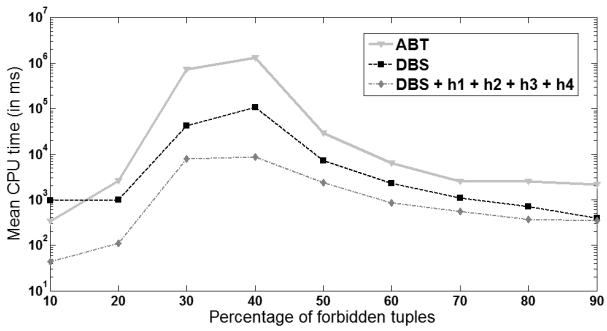


Figure 4: DisCSP constrained with 80% of the maximal constraint number.

100 instances) required by different DisCSP algorithms. Results are obtained with an Intel Core 2 duo 2.4 Ghz (4 Go of Ram).

On figures 3 and 4, we can observe that DBS is generally slightly faster than ABT for over-constrained and under-constrained DisCSP. For problems located in the transition phase, DBS outperforms ABT in terms of CPU time. Moreover, heuristics (see section 4.1) allow a reduction of the number of exchanged messages and CPU time. For DisCSP constrained with 80% of the maximal constraint number, during the transition phase, we stopped ABT after four hours of computation and we do not report it in the average.

Table 1 shows the results obtained with $< 20, 10, 0.20, 0.60 >$ DisCSP. We observe that the number of exchanged messages for ABT is lower compared to DBS. DBS requires less operations than ABT to process received backtrack messages: ABT computes all inconsistent *agent_view* subsets and sends, for each subset, a backtrack message (It requires high computation costs). Unlike ABT, DBS transmits a backtrack message to the faulty agent or an agent with a lower priority than the faulty agent (It requires less computation costs but more messages).

Following a similar protocol, another experiment using the following parameters $< 15, 10, 0.50, 0.50 >$ was run (Table 2). ABT solves DisCSP using 109 700 messages in 428 seconds. DBS without heuristic solves DisCSP using 170 137 messages in 20 seconds and DBS using heuristics 44 915 messages in 9 seconds. We can observe that the standard deviation is very important. This is caused by the topology of the problems. Indeed there are simple problems which can be solved within 100 ms and much harder problems (generated with the same parameters) which require more than 500 seconds to be solved.

5. CONCLUSION

In this paper, we have proposed a DisCSP resolution algorithm. Its main feature is the use of sessions to determine message context. Moreover, DBS does not add communication links during the solution search between agents which are not sharing constraints. Indeed, when an agent transmits a backtrack message to an agent $A_i \notin \Gamma^-$, no trace of A_i is kept after the message has been sent. The DBS completeness proof has been shown. A comparison between DBS and ABT, a classical algorithm often used as a reference, has been given.

Currently, DBS only solves DisCSP where one variable is assigned to each agent. We plan to improve our algorithm in order to solve DisCSP containing multiple variables per agent. A more detailed comparison of DBS with recent DisCSP algorithms proposed in the litterature is also considered as a perspective of our work.

6. REFERENCES

[1] C. Bessiere, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In M. Silaghi, editor, *IJCAI'01 workshop on Distributed Constraint Reasoning*, pages 9–16, Seattle WA, 2001.

[2] A. Doniec, R. Mandiau, S. Piechowiak, and S. Espié. Anticipation based on constraint processing in a multi-agent context. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 17:339–361, 2008.

[3] A. Doniec, S. Piechowiak, and R. Mandiau. A discsp solving algorithm based on sessions. In I. Russell and Z. Markov, editors, *FLAIRS'05 : Recent advances in artificial intelligence: Proceedings of the eighteenth International Florida Artificial Intelligence Research Society Conference*, pages 666–670, Menlo Park, California, may 2005.

[4] C. Lecoutre, L. Sais, and J. Vion. Using sat encodings to derive csp value ordering heuristics. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:169–186, 2007.

[5] P. Monier, S. Piechowiak, and R. Mandiau. Multi-agent reasoning based on distributed csp using sessions : Dbs. In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (accepted paper)*, 2009.

[6] M. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *American Association for AI National Conference (AAAI'00)*, pages 917–922, 2000.

[7] T. Tsuruta and T. Shintani. Scheduling meetings using distributed valued constraint satisfaction algorithm. In *14th European Conference on Artificial Intelligence (ECAI)*, pages 383–387, 2000.

[8] M. Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems.* Springer, 2000.

Train Driver Rescheduling using Task-Exchange Teams

David G.A. Mobach[a] Erwin J.W. Abbink[b] Pieter J. Fioole[b] Ramon M. Lentink[b]
Leo G. Kroon[b,c] Eddy H.T. van der Heijden[a] Niek J.E. Wijngaards[a]

[a]D-CIS Lab
Thales Research & Technology NL
P.O. Box 90, 2600 AB
Delft

[b]Netherlands Railways
NSR Logistics Innovation
P.O. Box 2025, 3500 HA
Utrecht

[c]Rotterdam School of Management
Erasmus University Rotterdam
P.O. Box 1738, 3000 DR
Rotterdam

{david.mobach,eddy.vanderheijden,niek.wijngaards}@icis.decis.nl
{erwin.abbink,pieterjan.fioole,ramon.lentink,leo.kroon}@ns.nl

ABSTRACT

Crew rescheduling in response to disruptions is a difficult problem, due to the additional (social) constraints imposed on human workforce. In the real-world domain of train driver rescheduling in the Netherlands, an actor-agent based approach is taken to (a) support human dispatchers and (b) accommodate individual train drivers' preferences. This paper outlines the task-exchange team-configuration process including the role of the various rescheduling constraints. The rescheduling approach is designed for operation in a real world environment: to this end, a number of heuristics are discussed that are currently being examined to optimize the solution finding process with respect to three dimensions: *performance*, *quality* and *clarity*. The heuristics have been implemented in a research system, supporting the full driver-agent population, working on real world data. This effort is an ongoing study on novel multi-agent approaches to crew rescheduling, and is the result of cooperation between Netherlands Railways and D-CIS Lab.

Categories and Subject Descriptors

I.2.8 [**ARTIFICIAL INTELLIGENCE**]: Problem Solving, Control Methods, and Search – *scheduling*; I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence – *multi-agent systems*; I.2.1 [**ARTIFICIAL INTELLIGENCE**]: Applications and Expert Systems – actor-agent systems; J.m [**COMPUTER APPLICATIONS**]: Miscellaneous – *transportation*

General Terms

Algorithms, Design, Economics.

Keywords

Distributed Systems, Experimental, Multi-agent planning, Crew rescheduling, Railway, Actor-Agent.

1. TRAIN DRIVER RESCHEDULING

Applied research on advanced autonomous systems in a real-world domain provides a stimulating environment to demonstrate 'state of the art' research results and address the encountered pragmatic and fundamental challenges. In this paper a research system is described for the complex task of rescheduling tasks of train drivers in response to disruptions. The work results from cooperation between Netherlands Railways and D-CIS Lab.

1.1 Problem Domain

The railway operations of Netherlands Railways (NS) are based on an extensive planning process. In the planning process, the timetable is planned first, consisting of the line system and the arrival and departure times of trains. Next, the rolling stock stage supplies each train with sufficient rolling stock. Finally, the crew scheduling stage supplies each train with a train driver and sufficient conductors, as well as their duties (scheduled tasks). The total number of NS train drivers is about 3000. Each day, ca. 1000 duties are carried out while, at any moment in time, the number of active duties is ca. 300. NS train drivers operate from 29 crew depots. Each day a driver carries out a number of *tasks*, which means that he/she operates a train on a trip from a certain start location and start time to a certain end location and end time. The trips of the trains are defined by the timetable. Other 'tasks' include shunting, standby (i.e., being a spare driver), passenger (en-route via a train to a driving task), meal-breaks and free time. The tasks of the drivers have been organized in a number of duties: the tasks carried out by a single driver on a single day.

Crew scheduling in the railway domain is a relatively new area of research, as indicated by Ernst et al. [3] in their staff scheduling and rostering survey. Other transportation domains encountered include airline crew scheduling and bus crew scheduling. Freling et al. argue in [4] that compared to bus crew scheduling, train crew scheduling differs in a number of ways: Train crews travel more often as passengers; Train crews can be scheduled relatively far from crew home bases; Delays are more critical due to the high degree of connectivity in the train network (*knock-on effect*). Furthermore, scheduling of (train) crews is subject to many restrictions and constraints, such as, e.g., crew workload laws and agreements, familiarity of crew with rolling stock types and (parts of) the rail network, and constraints regarding the tasks within a duty (e.g. presence of meal breaks, buffer times to allow for changing of trains). After the planning process has finished, the daily plans are carried out in the real-time operations. Plans have to be updated continuously to deal with delays of trains and larger disruptions of the railway infrastructure. Disruptions may be due to incidents, delays, a breakdown of infrastructure or rolling

stock. On the Dutch rail network, on average 10 disruptions of a route occur per day. Delays occur more frequently: On average 450 trains experience one or more delays (> 3 minutes) out of the approximately 5000 train services per day. These delays lead to the removal of on average 10 train services per day.

The NS timetable and rolling stock circulation are cyclic in nature. NS crew duties, however, do not have a cyclic nature and thus require different rescheduling approaches. An additional complicating issue in a disrupted situation is the fact that the exact duration of the disruption is usually not known. That is, the initial estimate of the duration of a disruption often turns out to be incorrect. Currently, the NS crew rescheduling process is carried out by dispatcher-teams operating from four regional control centres. This organization requires extensive communication between these centres since many trains and duties operate in more than one region.

Jespersen-Groth et al. discuss crew rescheduling as part of railway disruption management in [5], describing both the disruption management process and the directly involved organizations. Railway disruption management consists of the following processes: *timetable adjustment*, *rolling stock rescheduling*, and *crew rescheduling*. These processes are carried out sequentially; where failure of a process leads to backtracking. A major objective in the disruption management process is to minimize the number of affected passengers. In the crew rescheduling process, more specific objectives are balanced: *feasibility*, *operational costs*, and *stability* (i.e. minimize the number of modified duties). Jespersen-Groth et al. further mention the lack of computerized support for railway disruption management, and a case is made for the use of Operations Research techniques in the disruption management process.

In the past years, NS has successfully applied novel Operations Research models to significantly improve the crew scheduling process for which they received the Edelman Award 2008 [6]. The current methods and techniques are very useful for generating the initial daily schedules, yet their calculation time is multiple hours, making them unfit for in-time rescheduling purposes. The cooperation between NS and D-CIS Lab studies a (decentralised) actor-agent based application for rescheduling of train drivers to support human dispatchers.

1.2 Research System

The work presented in this paper has been implemented in a research system, using real-world timetable and rolling stock schedule data and driver duty data. The research system currently focuses on rescheduling train driver duties in real-time over the course of a single day. It is assumed that any timetable and rolling stock plan modifications to cope with disruptions have been implemented, and a new rolling stock plan is in place, to which the driver schedules must be adapted.

The main goal of the solution process is to ensure that all trains are assigned drivers, whereby the solution-finding process as well as the solutions found by the research system must perform well in three separate dimensions:

- *performance*: to prevent knock-on effects, solutions must be found quickly (i.e. in the same order of time as human dispatchers, preferably much better),
- *quality of solutions*: although not as critical as performance, the solutions found must minimize adverse effects of modifying duties (e.g. introducing overtime, idle time, etc.),
- *clarity of solutions*: as human dispatchers remain responsible for implementing rescheduling solutions, the research system must ensure that solutions are understandable (e.g. minimizing the number of duty modifications, involved drivers, etc.)

Optimizing the solution-finding process for all three dimensions simultaneously is not always possible: for example, decreasing calculation time will limit the number of solution alternatives that can be examined. Tradeoffs have to be made between these areas to ensure solutions are applicable in the real world.

The aim of the research system is to construct a reliable and scalable operational system which can run together with the NS' current rescheduling systems in place. The current version (January 2009) is able to find rescheduling solutions for relatively large disruptions (e.g. blockages lasting 1~2 hours) with a full-size driver-agent population, faster than human dispatchers can resolve a single disrupted duty (solution time in the order of minutes). In [11], more details on the implementation, performance and results are presented. The following sections describe the actor-agent approach taken, the task-exchange team-configuration process, the role of the various team formation heuristics, and the route-analyzing capabilities. The paper concludes with a discussion on the balance between solution quality and performance of the solution finding process, and ends with indications of future research possibilities.

2. Actor-Agent Approach

The main objective of the train-driver rescheduling application is to realize a decentralized (multi-agent based) application which provides solutions faster than human dispatchers. In addition, the solutions should preferably minimize changes to original duties.

The research system is designed according to the actor-agent paradigm [10], which explicitly recognizes both human actors and artificial agents as equivalent team members, each fulfilling their respective roles in order to reach the team objectives. The actor-agent based design process provides the system with several useful global system characteristics. First, the decentralized approach in which agents use local knowledge, world views, and interactions, contributes to an open system design. This openness facilitates easy reconfiguration and/or adaptation to changing system requirements. Second, combining humans and agents within the system design allows for integrating them at their appropriate abstraction levels: Human dispatchers at the strategic/management level, train drivers at the level of defining and guarding their personal interests, and their respective agents at the level of implementing the strategic/management decisions and resolving actual schedule conflicts.

2.1 Architecture

The following actors and agents are recognized in the research system (see Figure 1):

Dispatcher-actor: A (human) dispatcher at one of the regional control centres, who interacts with the rescheduling system via one dispatcher-agent.

Driver-actor: A (human) train driver who imposes constraints on the rescheduling process based on the preferences he/she may have with respect to performing his/her duties. These constraints

can be hard (e.g., familiarity with rolling stock types) or soft (e.g., preferences for certain lines). Each driver-actor is associated with one driver-agent.

Dispatcher-agent: Presents a monitoring and control interface on the rescheduling process to the dispatcher-actor.

Process-manager-agent: The process-manager-agent (PMA) manages the rescheduling process, and provides a contact point for the rescheduling subsystem to dispatcher-agents. The PMA coordinates the rescheduling process by communicating the disruption information and parameters to the driver-agent population. The PMA maintains status information of the negotiation process and informs the dispatcher-agents.

Driver-agent: Responsible for resolving conflicts arising in duties due to disruptions. Each driver-agent (DA) represents a specific driver-actor in the rescheduling process. Driver-agents are capable of forming task-exchange teams to resolve a scheduling problem and participating in multiples of such teams.

Route-analyzer-agent: Driver-agents interact with a route-analyzer-agent (RAA) to determine the impact of changes to their existing duties. The RAA interacts with network-agents.

Network-agent: A network-agent (NA) maintains an up-to-date view of the railways network, reflecting any changes in timetable and rolling stock due to disruptions. The NA processes queries from the route-analyzer-agent.

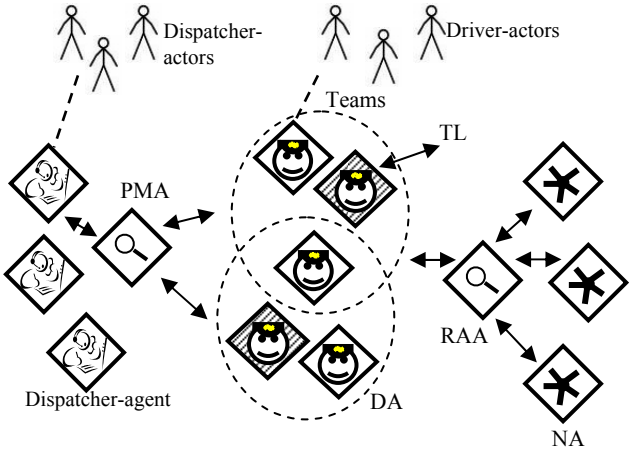


Figure 1: Actors, agents and their relations.

2.2 Agent-based Crew Scheduling

Agent-based crew-rescheduling is a relatively new area of research. To our knowledge, no research has been published on agent-based crew rescheduling applications in the railway domain. Shibghatullah et al. [8] propose an agent-based framework for bus crew scheduling including crew-reassignment. The paper provides an overview of the potential advantages of agent-based approaches (e.g., modelling individual preferences, more suited for partial, on-demand rescheduling), but lacks further details of the proposed framework. In [2], an agent-based approach to *airline operations recovery* is described, part of which concerns *crew recovery*. The architecture consists of specialized agents representing parts of the traditional Airline Operations Control Center organization. Similar to our approach, costs are assigned to various factors such as hotels and extra crew travel between the different operational bases.

The dynamics of the environment are an important factor in this domain: disruption estimates constantly change as more information is gathered; crews are constantly on the move between stations; knock-on effects, etc. De Weerdt et al. state in their overview of multi-agent planning [9] that although most researchers recognize the importance of dealing with changing environments, most planning approaches still assume fairly stable worlds. The authors mention contingency planning (plan for all contingencies that might occur) as a traditional approach of handling changes in the environment. As in many situations planning for all possible contingencies is not feasible, the authors argue that so-called *plan repair* approaches are more realistic: Detecting deviations from the original plan through monitoring, and adjust the plan as needed. Our approach can be viewed as plan repair, as we reactively resolve conflicts in driver duties upon the occurrence of disruptions. Furthermore, our research system uses distributed, local interactions to resolve duty conflicts. DesJardins et al. [1] present an overview of approaches in the field of distributed planning: Approaches are classified according to the properties they share with cooperative distributed planning (emphasis on forming a global (optimal) plan and negotiated distributed planning (emphasis on satisfying local goals). The authors argue that only recently research in this field addresses coping with dynamic, realistic environments. To cover this emerging work, the authors introduce the *distributed, continual planning* paradigm. This paradigm considers planning to be a dynamic ongoing process combining both planning & execution. Our work fits this paradigm, as the crew rescheduling process is performed in real-time and disruptions continuously require agents to revise their duties to cope with new circumstances.

3. TASK-EXCHANGE TEAMS

The basic principle underlying the solution process is that of *task exchange*. Each driver's schedule consists of a number of tasks. If in the event of a disruption a driver can no longer perform one or more tasks due to a schedule conflict, these tasks are taken over by another driver. In turn, this driver may have to hand over tasks which conflict with the newly accepted tasks to another driver. Using *cost functions*, the most favourable sequence of task exchanges can be selected as the solution for a team leader.

Schedule conflicts can be divided into two categories [5]: *time-based* and *location-based*. A location-conflict can occur when one or more tasks are removed from a schedule due to a disruption. A time-conflict occurs when due to a delay the arrival-time of the first task is later in time than the departure-time of the following task. Other disruptive circumstances (such as a train driver becoming unavailable to perform the remainder of his duty) are translated into these two types of conflicts. As a result of these conflicts, one or more tasks in the schedule become impossible for this driver to perform: These tasks have to be transferred to another driver. The process of finding task-exchanges which are feasible and desirable is performed by the driver-agent population.

In a task exchange, driver-agents assume the role of *team leader* or *team member*. In both cases, the aim of the agent is to resolve a *conflict*: In case of the team leader role, the conflict is due to modified tasks (e.g. as a result of a disruption, tasks may have

been removed or delayed). In case of the team member role, the conflict is due to tasks originating from another driver-agent.

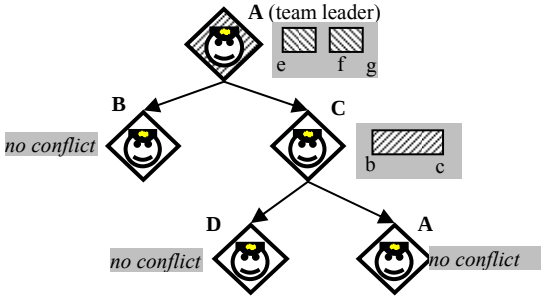


Figure 2 : Team configuration example.

A team leader starts a *recursive team extension* process by announcing its set of conflicting tasks to the driver-agent population. The team is subsequently extended with additional team members able to take over these conflicting tasks. Driver-agents can take over tasks either *unconditionally* (i.e. without creating a new conflict) or *conditionally* (i.e. creating a new conflict due to overlapping existing tasks). Driver-agents that respond conditionally announce their conflicts in a manner similar to a team leader. Driver-agents can participate multiple times in task-exchanges within the same team (and in other teams). This allows for teams to discover configurations in which driver-agents 'trade' tasks. Figure 2 shows an example of three team configurations: A-B, A-C-D and A-C-A. In this example, for each agent, the conflicting tasks that need to be resolved are shown: Agent C needs to resolve a conflict consisting of a single train driving task from station *b* to station *c*. The example also shows that agent A participates in the team both as team leader and as team member.

A sequence of task-exchanges is considered successful when the last task-exchange in the sequence does not result in a new conflict (or the conflict is sufficiently shifted forward in time to be resolved at a later point in time). At this point, the recursive team formation process is 'backtracked': Each team member selects the task exchange associated with the lowest cost. The protocol is explained in more detail in the following section.

3.1 Task Exchange Protocol

When a driver-agent is team leader or left with a new conflict as a result of accepting a task-exchange, an attempt is made to hand over the conflict to another driver-agent using the task-exchange protocol. In Figure 3, the message exchange is shown for a number of scenarios, discussed below. The commitment levels CL:1 and CL:2 are explained in section 3.2.

The protocol is initiated by a driver-agent in the role of team leader (TL): A *call* is issued by TL announcing the conflicting tasks to all driver-agents. Driver-agent DA1 has determined that it cannot participate in the task-exchange (i.e. either the exchange is infeasible or the associated costs are too high), and informs the TL by responding with a *not-interested* message.

Driver-agent DA2 on the other hand, is able to participate in the exchange, and indicates this by responding with an *interested* message. DA2 indicates that its interest is *interested conditional*, meaning that DA2 first has to exchange a new conflict to another driver-agent to solve the conflict from TL. DA2 now initiates its own task-exchange protocol, to which DA3 responds with an *interested* message. In this case, DA3 does not generate a new conflict. Instead, the costs for the task-exchange are calculated by DA3 and communicated to DA2 using a *quote* message.

When DA2 has received all quotes, the most favorable quote is selected (i.e. lowest quote value). In this example DA3's quote is selected and DA3 is informed using an *accept* message. DA3 confirms the acceptance using a *confirm* message. Based on the confirmed quote received from DA3, and the costs calculated for its own task-exchange with TL, DA2 can now report the total costs for the takeover to TL by sending a *quote* to TL. Similar to DA2 TL collects quotes, and confirms the quote from DA2.

The TL can finally decide on finishing the task-takeover process by issuing an *order* to DA2 (which is passed on to DA3): the returned *confirm* messages indicate that the team is ready to be *finalized*, which is the final message sent by TL. The protocol ends by DA3 returning a *team done* message to TL.

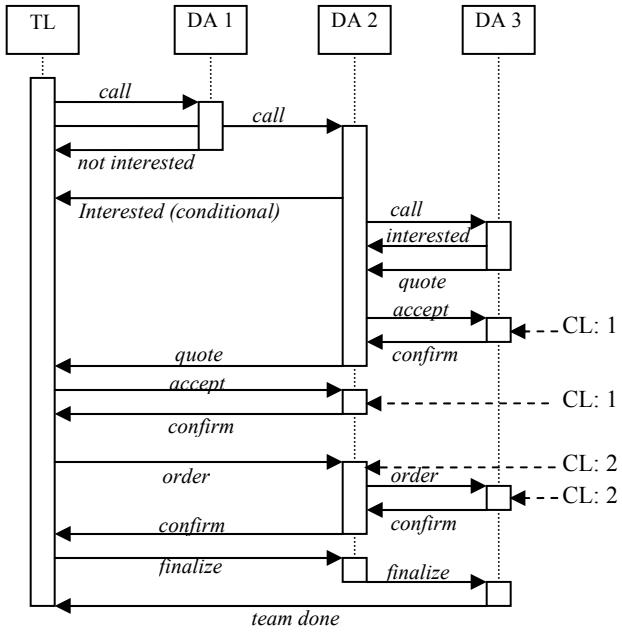


Figure 3: Task exchange protocol

3.2 Commitment Management

The design of the task-exchange process explicitly supports agents participating multiple times in task-exchange teams. To prevent driver-agents from committing in conflicting task-exchanges, a commitment management mechanism is included, based on commitment-levels [7]. The task-exchange protocol is divided into sections with *commitment levels*. Commitment level changes occur at specific points in the protocol (i.e. between these sections), as shown in Figure 3 (CL). At these points, a driver-agent applies a *commitment strategy* which specifies whether ongoing task-exchanges are allowed to proceed to the next level:

- **CL 1:** Upon receiving an *accept* message, a driver-agent needs to decide whether the task-exchange is still favourable, before confirming the acceptance.
- **CL 2:** Upon receiving an *order* message, a driver-agent must ensure that only non-overlapping task-exchanges are allowed to proceed, before confirming the order.

Every commitment level increase makes it more difficult for that driver-agent to *de*commit from this specific task exchange. CL:1

influences the behaviour of the team formation process, while in CL:2 consistency of the final solution is ensured.

3.3 Determining Duty Impact

Upon receiving a request for a task takeover, a driver-agent sends its (up-to-date) duty and the received conflict (i.e. tasks to be taken over) to the route-analyzer-agent (RAA). The RAA processes the request. In case the RAA has determined that the conflict can be integrated into the current driver's duty, the RAA's reply to the driver-agent contains two parts: a *delta* on the original duty (Δd) describing the required modifications to the duty, and a *new conflict* (NC) resulting from the modifications (i.e. tasks replaced with tasks from the conflict). The second part may contain zero or more tasks: in this case Δd did not lead to any tasks being replaced (i.e., the conflict is resolved without introducing another conflict, e.g. because the new tasks are replacing a break in the driver's duty).

In Figure 4 an example of this process is shown: The takeover of the conflict-tasks *e-f, f-g* by agent C results in a Δd consisting of tasks *b-e, e-f*, *f-g* and *g-c*. Due to the takeover, a new conflict is created consisting of task *b-c*, which may not include any part of the original conflict. For more information on the process of determining routes, see Section 5.

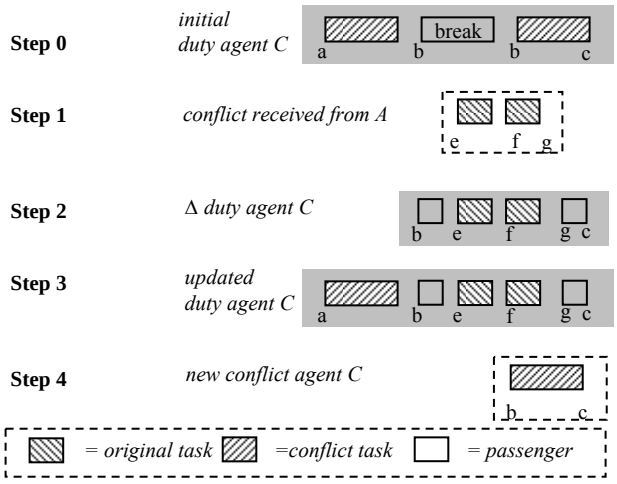


Figure 4: Example of duty impact.

4. Team Formation Heuristics

Due to the large number of driver-agents engaged in various roles in the team formation process and the fact that a short calculation time is an important success factor, heuristics are used to constrain the extension of teams with additional team members. The main goal behind the applied heuristics is to 'only let driver-agents participate that have a high probability of improving the solution found'. In this section, the applied heuristics are described: *interest determination, cost calculation*, *evaluating team score* and *decommitment determination*. These will be discussed in more detail below.

4.1 Interest Determination

Based upon the schedule impact determined earlier, a driver-agent decides whether it is *interested* in joining a team. Several domain-specific criteria are evaluated to determine a takeover desirability:

- If the impact of taking over a conflict results in a new conflict which is *larger* than the original, the agent is not interested.
- Taking over tasks from a driver-agent may only result in introducing a new conflict if the tasks in the new conflict are situated *later* in time.
- If the new conflict introduced by taking over tasks from a driver-agent consists of the *same set* (or superset) of tasks introduced by this agent elsewhere in the team, the agent will not be interested. This prevents repeating evaluation of sequences of task-exchanges that are already being evaluated.

4.2 Costs Calculation

A driver-agent interested in taking over tasks must determine the costs associated with this takeover. The cost function is non-decreasing and assigns costs to the following elements:

- Amount of overtime introduced by extending a schedule past the original end time;
- Affecting meal break;
- Use of spare drivers. Spare drivers are preferred in case of larger disruptions, and should not be used for resolving small/simple conflicts;
- Size of team (e.g. preferring recurring team members);
- Individual train driver preferences (i.e., modifiers to the above elements).

The relative weights of the cost elements can be varied to prioritize different aspects of the solutions that are found (e.g. prevent the use of spare drivers by increasing their relative cost). The costs are subsequently compared to costs of other exchanges using a scoreboard mechanism, described below.

4.3 Team-Configuration Scoreboard

Upon starting a task exchange process, each team leader publishes a *scoreboard*, which can be used by team members to inform other driver-agents of the progress within a task-exchange team. The scoreboard is based on the principle that every driver-agent in a configuration of a team has knowledge of the 'cost' of the exchange configuration (the task exchanges leading up to the task-exchange this driver-agent is currently examining).

The moment a driver-agent has determined that a conflict can be resolved without generating a new conflict, it publishes the value of the current solution (i.e., cost of one complete (possible) configuration) on the scoreboard. All driver-agents can access the scoreboard and examine the costs of the solutions which are already found. Driver-agents can use the values on the scoreboard to decide whether to continue a task-exchange or not. If the cost of the current partial solution is already the same or higher than the scoreboard value, the driver-agent will terminate its involvement in this branch of the task-exchange.

In Figure 5, an example is shown: First, driver-agent B determines that it can solve the conflict of A(TL). It finds that the scoreboard is still empty, and updates it with the calculated costs (arrow 1). Subsequently, C finds that in order to solve the teamleader's conflict, it has to introduce a new conflict. Before continuing, it checks the scoreboard to determine if the current costs still are an improvement. In this case C continues. Agent D determines that it

can solve the conflict of B without introducing a new conflict. It also determines that the cost of the solution improves the current score on the scoreboard. D updates the scoreboard (arrow 2). Finally, agent A(DA) aborts, as it has determined that its (partial) solution does not improve the team's current score.

The scoreboard mechanism ensures that only solution alternatives are evaluated which improve the currently found solution(s). Note that this mechanism depends on the cost function to accurately discriminate between 'better', 'similar' and 'worse' solutions.

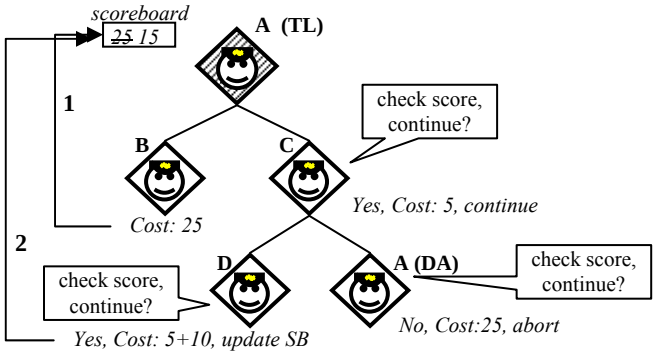


Figure 5 : Scoreboard example

5. ANALYZING ROUTES

In this section, the agents involved in determining the feasibility and impact of schedule changes on driver schedules are discussed. This rather computationally expensive functionality is deliberately separated from the driver-agents task-exchange capabilities to enhance performance and ensure cleaner division of responsibilities. The route-analyzer-agent (RAA) is the central point of contact for the driver-agents; distributed network-agents (NA) support the RAA.

5.1 Route-Analyzer-Agent

Requests for route calculations from the driver-agents are handled by a RAA. A request consists of a duty and a conflict. The duty is the current duty of the requesting driver-agent. The conflict consists of one or more tasks to take over from another driver-agent. The answer returned by the RAA can either be *feasible*, *feasible conditional* or *not feasible*. Feasible indicates that it is possible to add the conflict to the duty of the driver-agent without introducing a new conflict. In addition to the conflict one or more passenger tasks (trips to/from the conflict) may be added to the duty. If it is possible to add the conflict to the duty of the driver-agent, yet a new conflict is introduced in the process, the answer is *feasible conditional*. In this case, as mentioned earlier in section 3.3, the new conflict may not contain any tasks contained in the old conflict. When it is not possible to add the conflict to the duty of the driver-agent at all, the answer will be *not feasible*.

The RAA attempts to determine the correct answer to a request without having to take into account the detailed current state of the rail network. To this end, the RAA performs three steps. First, a check is performed to determine if a negative answer can be given quickly. After this the request is compared to a history of previously received requests. If no answer is found the requests are distributed over the available NAs for detailed examination. These steps are described in more detail below.

Step 1: Sanity Check: In a large number of cases (~50%) it can be easily determined that a request is not feasible. Consider for example a request where a driver-agent's current location is Rotterdam and tasks to take over are in the vicinity of Groningen, and take place within 15 minutes. The distance between Rotterdam and Groningen is more than 200 kilometers: this clearly is an impossible takeover. The RAA uses an origin-destination matrix with lower bounds on the travel time between all stations. These lower bounds are static and calculated beforehand by taking into account only the tracks in the network and maximum driving speeds of the available train-units. If the lower bound is larger than the available time we know it is not possible to find any route.

Step 2: Request History: The RAA retains all calculated routes in memory. If the same request is received more than once (possibly by different driver-agents), the answer is retrieved from this history (~4-5% of the remaining requests). Furthermore, if a previous request with a wider time-interval for the same route resulted in *not feasible*, the route-analyzer-agent can conclude the current request also results in a *not feasible* answer. This history is reset when changes on the rail network.

Step 3: Send to network-agent: If no relevant requests are found in the request history, the RAA sends the request to one of the NAs and returns the thus obtained answer to the driver-agent.

Table 1: Example duty for driver-agent.

Departure	Destination	Departure time	Arrival time
Rotterdam	Utrecht	7:45	8:24
Utrecht	Woerden	9:08	9:22
Woerden	Rotterdam	9:32	10:08
Rotterdam	Maassluis	10:58	11:21
Maassluis	Rotterdam	11:29	11:53
Rotterdam	Eindhoven	11:47	13:00
Eindhoven	Rotterdam	13:32	14:42

As an illustration, consider a driver-agent with a duty as shown in Table 1. This driver-agent wants to know if he can take over a trip from another agent. This trip departs at 10:51 from Den Haag and arrives at 11:10 in Gouda (not shown in Table 1; this driver-agent is not a team leader, i.e. not directly affected). The driver-agent sends this request to the route-analyzer-agent when the actual time is 9:00. The RAA determines that the driver-agent's current location is Utrecht. The minimal time to get from Utrecht to Den Haag is 35 minutes. In this case the driver-agent has 111 minutes available (i.e., from 9:00 tot 10:51) so a route might exist. Similarly, the RAA concludes that a route from Gouda back to Rotterdam could also exist. The RAA concludes that further examination of the request is necessary.

When all NAs are unavailable the RAA maintains a priority queue of requests to send to a network-agent. For each request the RAA assigns a *prediction-value* and keeps the queue sorted according to this value. When a NA becomes available the RAA sends the request with the best prediction-value. The prediction-value represents an expectation by the RAA of how well the conflict can be fitted into the duty. This is the weighted sum of:

- Train driver duty task lengths in the conflict time interval.

- Lower bounds on travel time from current location to start of conflict, and lower bounds on travel time from end of conflict to base.

These items are determined by an initial analysis of the outcome of a couple of thousand requests; where the weights were determined by regression analysis. For example, it is straight-forward to understand that if a driver-agent has a lot of work within the time-interval of the conflict he will surely have to send out a new conflict if he takes over the current conflict in his request. Similarly agents that are located closer to a conflict are more likely to take over the conflict in an efficient way. An important factor contributing to the success of this scoreboard mechanism is a *first value* being published as soon as possible. Sorting the requests this way helps to find good solutions more quickly. Once a good solution is found in a team the scoreboard is seeded with this first score, activating the score-board mechanism (see section 4.3).

5.2 Network-Agents

The NA maintains knowledge of the current time-table, including all disruptions and delays. In the actor-agent community at least one NA has to exist, but possibly more. NAs can easily be distributed over multiple platforms (no cooperation is necessary and the data can be replicated). Based on this time-table the NA can determine, by using a shortest-path algorithm, if a route between two stations exists, and if so, which route. The objectives in this process are to (i) maintain as much of the original duty of the driver-agent as possible, and (ii) to arrive at the destination (i.e. the conflict's location) as soon as possible. To illustrate this consider again the example of Table 1, continued in Table 2.

Table 2: New duty for driver-agent

Departure	Destination	Departure time	Arrival time
Rotterdam	Utrecht	7:45	8:24
Utrecht	Woerden	9:08	9:22
Woerden	Rotterdam	9:32	10:08
Rotterdam	**Den Haag**	**10:14**	**10:40**
Den Haag	Gouda	10:51	11:10
Gouda	**Rotterdam**	**11:16**	**11:38**
Rotterdam	Eindhoven	11:47	13:00
Eindhoven	Rotterdam	13:32	14:42

The NA attempts to find a route from Utrecht to Den Haag between 9:00 and 10:51. The fastest way to get to Den Haag is a direct connection which departs from Utrecht at 9:03 and arrives in Den Haag at 9:51. This means a route actually does exist. But there also exists a better alternative for this driver-agent: The NA also finds a direct connection between Rotterdam and Den Haag departing at 10:14 and arriving at 10:40. This way the train driver can still perform the tasks of its own duty until 10:08 and still be in time for the task-exchange trip.

Finally the NA is able to find a trip from Gouda to Rotterdam, so the driver-agent can also perform the last two tasks from its own duty. This means, if the driver-agent wants to take over the requested trip, its new duty will be as shown in Table 2. The underlined trip is taken over by this driver-agent, while for the **bold** trips the driver rides the train as a passenger. In this case the driver-actor can no longer perform the task from Rotterdam to Maassluis (10:38-11:21) and from Maasluis to Rotterdam (11:29 – 11:53). This is the *feasible conditional* answer the NA will return to the RAA, which forwards this to the driver-agent.

Note that this change to the driver-agent's duty also means that the 50 minute break in Rotterdam from 10:08 until 10:58 no longer exists. The driver-agent will take this into account when examining the schedule changes (Δd); the NA only assesses if a route is *possible*, not whether the route is *desirable*.

In case the conflict can be fitted into the duty (as specified in the request received by the NA), the returned answer consists of the necessary duty adjustments and a set of tasks which can no longer be performed. This set of tasks can be empty; in that case the conflict can be fitted into the duty without introducing a new conflict. When the conflict cannot be fitted into the duty ('failure'), the NA returns no duty adjustments nor a set of tasks.

6. DISCUSSION

In the real-world domain of train driver rescheduling in the Netherlands, an actor-agent based approach is taken to (a) support human dispatchers and (b) accommodate individual train drivers' preferences. The previous sections have provided an outline of the task-exchange team-configuration process including the heuristics used to regulate the process and the role of the various rescheduling constraints. Summarizing, two multi-agent sub-systems have been co-developed in the research system: train-driver rescheduler and route-analyzer.

The research system aims to find a balance between optimizing for performance, quality and clarity of solutions. With respect to performance and quality, good results have already been obtained (see [11]). Human dispatchers are currently being consulted to assess the clarity of the solutions. An important factor contributing to the clarity of the solutions is the fact that the basic principle of task-exchange teams is straightforward and resembles in many ways the rescheduling process human dispatchers use.

Although the team formation process supports multiple simultaneous team formations, some coordination of teams remains necessary: when dealing with teams resulting from a single disruption, teams are highly likely to consist of the same driver agents, resulting in a high probability of decommitment (and thereby team re-formation processes). A relatively simple way to reduce the overhead of decommitments is running teams *sequentially* while allowing teams related to other (geographically spaced) disruptions to run in parallel.

The scoreboard mechanism must also take decommitment behavior into account: as it is possible for an agent that has previously posted a score to decommit from the solution related to this (now possibly invalid) score, the (use of the) scoreboard can be configured in the following ways:

- Increase the number of scores that may be posted on the scoreboard: this reduces the chance of all scores on the scoreboard to be decommitted.
- Increase the penalty on decommitting if an agent has posted a score on the scoreboard.
- Ensure that an agent that posts a score in a team decommits from all other teams.

In this paper a number of optimizations are discussed that are currently being researched and implemented. These optimizations are aimed at improving the performance of the solution finding

process and quality as well as clarity of the solutions found. With respect to performance, the main factor impacting performance is the route finding process. In addition to the mechanisms described in this paper, the route finding capacity of the system can be scaled up relatively easy by adding more network-agents and additional supporting hardware. With respect to solution quality, the effectiveness of the heuristics influencing the team formation process is the determining factor. The heuristics described in this paper are currently being evaluated and improved as part of ongoing research. Current research focuses on comparing results of realistic scenarios against actual solutions determined by human dispatchers, in order to assess and improve our heuristics.

7. FUTURE RESEARCH

One of the current research efforts focuses on extending the research system for train-driver rescheduling with agents representing train stations, thereby introducing a different view on the railway network than driver-agents, who are mainly concerned with their own duties. Station-agents aim to improve the robustness of the resulting driver duties against future disruptions. When facing new disruptions, a robust set of duties requires little modifications to deal with the disruptions. The proposed extension is to measure such robustness by monitoring the 'inventory' of train drivers at interesting (i.e. large) stations during the day: the *station fitness*.

Another extension is to acquire more understanding of the (emergent) behavior of the research system, e.g., by analyzing communication patterns. One potential advantage is to improve the route-analyzer-agent further in handling requests without involving network-agents. Another potential advantage is to forward messages faster to relevant driver-agents. It is important, though, to avoid dependencies on structures in the original crew schedule, as these may be changed by NS over time.

An important factor in the formation of teams is to ensure that driver-agents decommit from less promising task-exchanges at the right moment. Decommitting too early will mean that potential solutions will be lost, decommitting too late will impact performance as entire task-takeover structures are 'rolled back' and need to be redone. This tradeoff between performance and solution quality is (also) subject of ongoing research.

Future extensions concern handling stochastics (e.g., uncertain duration of blockages), and partially accepting conflicts (without breaking the properties of the protocol).

8. ACKNOWLEDGMENTS

The authors express their gratitude to NS and Cor Baars (DNV / Cibit) for starting this project. The following D-CIS Lab colleagues provided valuable contributions: Hilbrandt van Boven, Pascal Hoetmer, Michel Oey, Reinier Timmer, Louis Oudhuis, Martijn Broos, Sorin Iacob, Thomas Hood, and Kees Nieuwenhuis. The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project (www.icis.decis.nl), supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. The ICIS project is hosted by D-CIS Lab (www.decis.nl), the open research partnership of Thales Nederland, the Delft University of Technology, the University of Amsterdam and the Netherlands Organisation for Applied Scientific Research (TNO).

9. REFERENCES

[1] desJardins, M.E., Durfee, E.H., Ortiz, C.L., and Wolverton, M.J. (1999). A Survey of Research in Distributed, Continual Planning, *AI Magazine*, **4**, pp. 13-22.

[2] Castro, A.J.M., Oliveira, E. (2007), Using Specialized Agents in a Distributed MAS to Solve Airline Operations Problems: a Case Study. In: *Proceedings of the 2007 IEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, pp. 473-476, IEEE Computer Society

[3] Ernst, A.T., Jiang, H., Krishnamoorthy, M., and Sier, D. (2004), Staff Scheduling and Rostering: A Review of Applications, Methods, and Models. In: *Eur. Jnl of Operational Research*, **153**, pp. 3-27.

[4] Freling, R., Lentink, R.M., Odijk, M.A. (2000), Scheduling Train Crews: a Case Study for the Dutch Railways. In: *Proceedings of CASPT 2000*.

[5] Jespersen-Groth, J., Pothoff, D., Clausen, J., Huisman, D., Kroon, L., Maróti, G., and Nyhave Nielsen, M. (2007), *Disruption Management in Passenger Railway Transportation*. Report EI2007-05, Econometric Institute, Erasmus University Rotterdam (2007), 35 pages. (Submitted to Computers & Operations Research in January 2007).

[6] Kroon, L., Huisman, D., Abbink, E., Fioole, P.J., Fischetti, M., Maróti, G., Schrijver, L., Steenbeek, A., Ybema, R. (2009). The New Dutch Timetable: The OR Revolution. In: *Interfaces,* **39**(1), pp. 6-17.

[7] Sandholm, T. and Lesser, V. (2002). Leveled-commitment contracting: a backtracking instrument for multiagent systems. *AI Mag.* **23**, 3 (Sep. 2002), 89-100.

[8] Shibghatullah, A.S., Eldabi, T., Rzevski, G. (2006), A Framework for Crew Scheduling Management System Using Multi-Agents System. In: *28th Int. Conf. on Information Technology Interfaces (ITI 2006)*, Cavtat, Croatia.

[9] de Weerdt, M., ter Mors, A., and Witteveen, C. (2005), Multi-agent Planning: An introduction to planning and coordination. In: *Handouts of the EASSS*, pp. 1-32.

[10] Wijngaards, N., Kempen, M., Smit, A., and Nieuwenhuis, K. (2006), Towards Sustained Team Effectiveness. In: Lindemann, G., et al. (Eds.), Selected revised papers from the workshops on *Norms and Institutions for Regulated Multi-Agent Systems* (ANIREM) and *Organizations and Organization Oriented Programming* at AAMAS'05, LNCS, Springer Verlag, **3913**, pp. 33-45.

[11] Abbink, E.J.W., Mobach, D.G.A., Fioole, P.J., Kroon, L.G., van der Heijden, E.H.T., Wijngaards, N.J.E. (2009), Actor-Agent Application for Train Driver Rescheduling. In: Proceedings of AAMAS 2009, *in press*.

A Multi-Agent Learning Approach for the Multi-Mode Resource-Constrained Project Scheduling Problem

Tony Wauters, Katja Verbeeck, Greet Vanden Berghe
Vakgroep IT
KaHo Sint-Lieven
Gebroeders Desmetstraat 1
B-9000 Gent, Belgium
{FirstName.LastName}@kahosl.be

Patrick De Causmaecker
Faculty of Sciences, Department of Computerscience
K.U. Leuven Campus Kortrijk
Etienne Sabbelaan 53
B-8500 Kortrijk, Belgium
patrick.decausmaecker@kuleuven-kortrijk.be

ABSTRACT

This paper introduces a novel approach for solving the multi-mode resource-constrained project scheduling problem (MRCPSP), in which multiple execution modes are available for each of the activities of the project. The new approach applies simple agent learning devices, i.e. learning automata, to construct the project schedules. We present some comparative results, to show that our decentralized method can easily compete with the best performing algorithms for the MRCPSP.

1. INTRODUCTION

In the last few decades, the resource-constrained project scheduling problem (RCPSP) has become a popular subject in operations research. It consists of scheduling the activities from a project by respecting the resource requirements and precedence relations between the activities.
The MRCPSP is a generalized version of the RCPSP, where each activity can be performed in one out of a set of modes, with a specific activity duration and resource requirements (e.g. 2 people each with their own shovel need 6 days to dig a pit, while 4 people each with their own shovel and one additional wheelbarrow need only 2 days). A comprehensive survey of the project scheduling problem can be found in [3]. The latter paper presents a unifying notation, a model, a classification scheme, i.e. a description of the resource environment, the activity characteristics, and the objective function, respectively. The notation is similar to machine scheduling and allows to classify the most important models. It also introduces some exact and heuristic methods for both single and multi-mode problems. In [6] Herroelen et al. discuss the problem and its practical relevance. Kolisch and Hartmann [13] provide an update of their survey that was first published in 2000. They summarize and categorize a large number of heuristics that have recently been proposed in the literature together with some detailed comparitive results. The RCPSP is shown to be an NP-hard optimization problem [1], thus so is the MRCPSP, because it is a generalisation of the RCPSP [11]. In order to produce good quality solutions in a short amount of time, different kinds of (meta-)heuristics have been proposed to address the problem. A tabu search metaheuristic is applied in [5] for solving the MRCPSP with generalized precedence relations. In [4, 7, 16, 17] a genetic algorithm is presented for the MRCPSP.

Agent-based approaches have also been succesfully applied to the MRCPSP. In [10] two types of agents (basic and enhanced agents) are used, together with a set of different priority rules. The two agent types differ by the feasible execution mode that is selected for each resource. A basic agent, which is purely reactive, chooses the first feasible execution mode that it finds. In contrast, an enhanced agent deliberates the mode selection according to several rules. In [9] a number of agents, each representing a different optimization algorithm including local search, tabu search, as well as several specialized heuristics, have been used to work on a population of solutions in parallel on different computers. In [8] a population learning algorithm is presented for solving both the single and the multi-mode rcpsp.
In this paper we present a novel multi-agent based approach, fundamentally different from these other agent based approaches. It is based on a graph representation in which the nodes are agents representing activities. Each agent is responsible for one activity. The agents make use of two learning devices, i.e. learning automata (LA) to construct a schedule. We use a smart coupling mechanism of rewards to learn both the order of the activities and the modes at the same time. Further on our approach is inspired by theoretical results that show how interconnected LA devices are able to find attractor points in MDP's and Markov Games [18, 19]. Altough the activity-on-node model for the MRCPSP problem as we consider here does not satisfy the Markov property as was assumed in [18, 19], good results are produced.

This paper is structured as follows. Section 2 gives a brief problem description, followed by our model and multi-agent learning algorithm in Section 3. In Section 4 we present some computational experiments and comparative results. Finally in Section 5 we draw conclusions and discuss future work.

2. PROBLEM FORMULATION

The MRCPSP can be formulated as follows. A project has N activities $a_1, \ldots, a_N; a_i \in A$, with A the set of activ-

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

ities, where each activity can be performed in one out of a set of K modes $m_{i1}, \ldots, m_{iK}; m_{ij} \in M_i$, with M_i the set of modes for activity i. Each mode corresponds to a specific activity duration $d_{m_{ij}}$ and resource requirements ($ren^l_{m_{ij}}$ for the renewable resources and $nren^l_{m_{ij}}$ for the non-renewable resources). The dummy start and end activities 1 and N have zero duration and zero resource usage. These activities have to be scheduled according to their finish-start precedence relations. Any activity i has a set P_i of activities as its predeccessors, and also a set S_i of activites as its successors. The project has some renewable and non-renewable resources available each with their own availability. For each renewable resource l the total availability ren^l is constant throughout the problem horizon, while the non-renewable resources have a limited total usage $nren^l$.

In the MRCPSP, the objective is to find an activity order - mode combination that produces a schedule that minimises the project makespan and is subject to two hard constraints: 1) an activity should not be scheduled until all its predecessors have finished (precedence constraint) and 2) the number of assignments of a resource at any time should not be larger than the availability of that resource (resource constraint). With s_i the start time and d_i the duration of activity i we can also formulate the problem as follows:

$$min\ s_N \tag{1}$$

$$s.t.$$

$$\sum_{j=1}^{K} m_{ij} = 1 \qquad i = 1, \ldots, N \tag{2}$$

$$s_p + d_p \leq s_i \qquad \forall_i; s_p \in P_i \tag{3}$$

$$d_i = \sum_{j=1}^{K} m_{ij}\ d_{m_{ij}} \qquad i = 1, \ldots, N \tag{4}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K} m_{ij}\ nren^l_{m_{ij}} \leq nren^l \qquad \forall_l \tag{5}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K} m_{ij}\ ren^l_{m_{ij}}\ e_{it} \leq ren^l \qquad \forall_l; t = s_1, \ldots, s_N \tag{6}$$

$$\left\{ \begin{matrix} e_{it}=1 & if\ s_i \leq t < s_i + d_i \\ e_{it}=0 & o.w. \end{matrix} \right\} \qquad \forall_{i,t} \tag{7}$$

$$m_{ij}, e_{it} \in \{0, 1\} \qquad \forall_{i,j,t} \tag{8}$$

$$d_{m_{ij}}, s_i \in \mathbb{N} \qquad \forall_{i,j} \tag{9}$$

$$s_0 = 0 \tag{10}$$

3. THE MULTI-AGENT LEARNING APPROACH

The (M)RCPSP can be presented with an activity-on-node diagram. It uses a graph to show the precedence relations between the activities. In Figure 1 we see an example of a project with 7 activities according to the problem description in Section 2 (1 and 7 are dummy activities) and their relations. That respresentation is the starting point, for the multi-agent based learning algorithm we developed.

Our goal is to create an activity order list and a mapping from activities to modes, which can later be used to construct a schedule. The activity order list is a permutation of all the activities, and determines in which order the schedule construction algorithm handles the activities. The mode

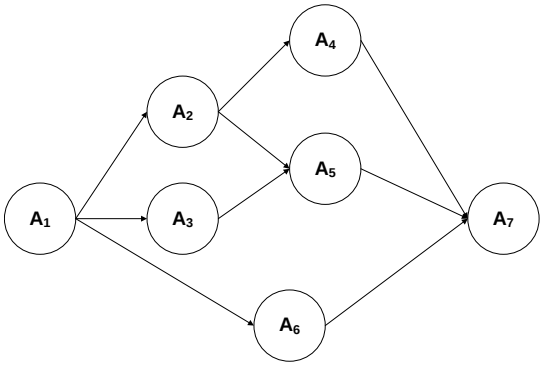


Figure 1: An example of an activity-on-node diagram for a project with 7 activities.

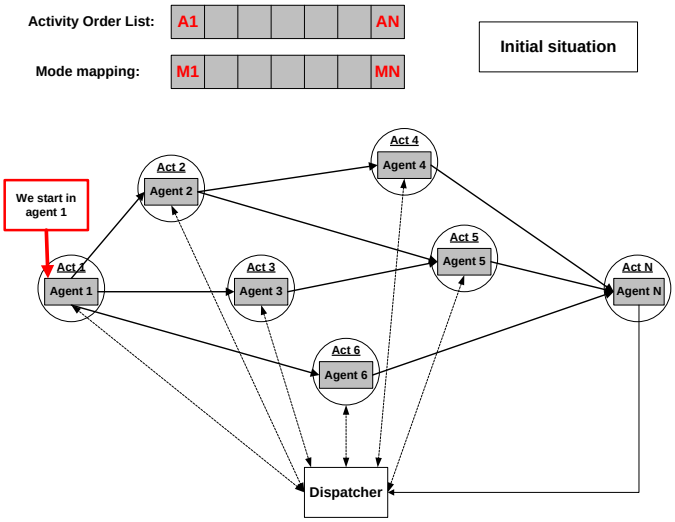


Figure 2: Initial situation, one agent in every activity node + one extra dispatching agent.

mapping determines in which mode each activity will be executed. We start by placing an agent in every activity node. Further on we add an extra dispatching agent (dispatcher) which is needed by our algorithm to construct schedules that respect the two hard constraints. In contrast to the other agents, the dispatcher does not represent an activity and only chooses an other agent to hand over the control. This initial situation is presented in Figure 2.
The main idea of the algorithm is to enable every agent to learn which decisions to make, concerning:

1. in which order to visit its successors, and
2. in which mode the activity needs to be performed.

The algorithm works as follows: we start in the situation as in Figure 2, we give the control to agent 1, this agent chooses an order to visit its successors and picks the first agent from this order ($Agent_{next}$). Its activity is already in the activity list so it does not need to choose a mode. Now the control is given to $Agent_{current} \Leftarrow Agent_{next}$, which also decides upon an order to visit its successors (e.g. A_2 chooses order $|A_5, A_4|$, so it will first take A_5 and then A_4) and takes the first agent from this order ($Agent_{next}$). $Agent_{current}$ has

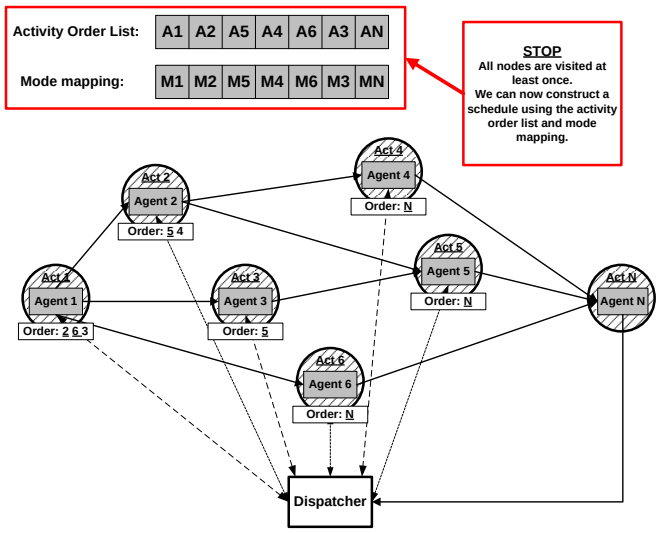


Figure 3: Final situation, all agents have been visited at least once.

not been visited before. Consequently the activity it represents is added to the activty order list, and the agent also chooses a mode which is added to the mode mapping. This process is continued until the agent in the last dummy node is visited. This node is special in the sense that its agent does not need to choose an activity order or a mode, but always forwards the control to the dispatcher. The dispatcher has a certain probability ($Pr_{RandToVisited}$) to choose a random eligible agent from the list of already visited agents, otherwise it chooses a random eligible unvisited agent. An agent is eligible when all the predecessors of the activity it represents have been visited. Note that this simple random dispatcher strategy can be changed into something else (e.g. a heuristic strategy). These steps are carried out subsequently until all agents have been visited at least once.

In addition to all the previous, the agents behave stochastically. At any time they can give the control, with a small probability Pr_{ToDisp}, to the dispatcher. This makes it possible to naturally consider all possible activity-order permutations and hence all the possible schedules.

Now we can construct a schedule with the activity order and mode mapping with a serial schedule generation scheme that uses a standard heuristic method for RCPSP (see [12] for details).

3.1 Algorithm

In this section we present the algorithm behind our approach in a more formal way by using pseudo code. In Algorithm 1 the global control of the algorithm for constructing an Activity Order List and Mode Mapping is presented. From there the control is given to individual agents which use Algorithm 2. The latter returns the next agent to visit ($Agent_{next}$) which is given control by the global control.
For clarity we left out the implementation of the method to determine if an Agent is eligble.

The method described above for constructing a schedule can now be used in an iterative way. We use the schedule's quality (makespan) for the agents to learn the actions to take. We apply some simple learning automata devices

Algorithm 1 Global Control

Input: Project data and Algorithm parameters
Output: A precedence constraint feasable schedule
  initialize $ActivityOrderList$ and $ModeMappingList$
  $Agent_{current} \Leftarrow Agent_1$
  while Not all agents visited **do**
    give the control to $Agent_{current}$
    $Agent_{next}$ determined by $Agent_{current}$ using Algo. 2
    if $Agent_{next}$ is eligible **then**
      $Agent_{current} \Leftarrow Agent_{next}$
    else
      $Agent_{current} \Leftarrow Dispatcher$
    end if
  end while
  $Schedule \Leftarrow$ construct a schedule using the obtained $ActivityOrderList$ and $ModeMappingList$
  return $Schedule$

Algorithm 2 Single Agent Control

Input: $ActivityOrderList$, $ModeMappingList$
Output: $Agent_{next}$
  rand $\Leftarrow$ random number between 0 and 1
  if $(rand < Pr_{ToDisp})$ or (this is $Agent_N$) **then**
    $Agent_{next} \Leftarrow Dispatcher$
  else
    if $Agent_{current}$ not yet visited **then**
      add $Agent_{current}$ to the $ActivityOrderList$
      $Mode \Leftarrow$ chooseMode() using Mode LA
      add the $Mode$ to the $ModeMappingList$
      $Order \Leftarrow$ chooseOrder() using Order LA
      $Agent_{next} \Leftarrow$ first Agent in $Order$
    else
      $Agent_{next} \Leftarrow$ next Agent in $Order$
    end if
  end if
  return $Agent_{next}$

which we will describe in the next Section.

3.2 Learning Automata

Learning Automata are simple reinforcement learners originally introduced to study human and animal behavior. The objective of an automaton is to learn an optimal action, based on past actions and environmental feedback. Formally the automaton is described by a quadruple $\{A, \beta, p, U\}$, where $A = \{a_1, \ldots, a_r\}$ is the set of possible actions the automaton can perform, p is the probability distribution over these actions, β is a random variable between 0 and 1 representing the evironmental response, and U is a learning scheme used to update p.

A single automaton is connected in a feedback loop with its environment. Actions chosen by the automaton are given as input to the environment and the environmental response to these actions serves as input to the automaton. Several automaton update schemes with different properties have been studied. Important examples of linear update schemes are linear reward-penalty, linear reward-inaction and linear reward-ϵ-penalty. The philosophy of these schemes is essentially to increase the probability to select an action when it results in a success and to decrease it when the response is a failure. The general algorithm is given by:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \alpha_{reward}(1-\beta(t))(1-p_m(t)) \\ &\quad - \alpha_{penalty}\beta(t)p_m(t) \qquad (11) \\ &\quad \text{if } a_m \text{ is the action taken at time } t \\ p_j(t+1) &= p_j(t) - \alpha_{reward}(1-\beta(t))p_j(t) \\ &\quad + \alpha_{penalty}\beta(t)[(r-1)^{-1} - p_j(t)] \qquad (12) \\ &\quad \text{if } a_j \neq a_m \end{aligned}$$

The constants α_{reward} en $\alpha_{penalty}$ are the reward and penalty parameters respectively. When $\alpha_{reward} = \alpha_{penalty}$, the algorithm is referred to as linear reward-penalty (L_{R-P}), when $\alpha_{penalty} = 0$, it is referred to as linear reward-inaction (L_{R-I}) and when $\alpha_{penalty}$ is small compared to α_{reward} it is called linear reward-ϵ-penalty ($L_{R-\epsilon P}$).

A motivation for using learning automata is that nice theoretical convergence properties are proven to hold in both single and multi automata environments. One of the principal contributions of LA theory is that a set of decentralized learning automata using the reward-inaction update scheme is able to control a finite Markov Chain with unknown transition probabilities and rewards. Recently this result was extended to the framework of Markov Games, a straightforward extension of single-agent markov decision problems (MDP's) to distributed multi-agent decision problems [15].

3.3 LA for the MRCPSP

For learning the activity order and the best modes we applied the (L_{R-I}) method because of its ϵ-optimality property in all stationary environments. The learning rate (reward parameter) that is used for learning the activity order, and the one that is used for learning the mode are named LRO and LRM. The application of the reinforcement will be presented in what follows.

After a schedule was constructed, we update all the learning automata using the following reinforcements. If the makespan of the constructed schedule was:

- Better: reinforcement = 1
- Equal: reinforcement = r_{eq} ($r_{eq} \in [0, 1]$)
- Worse: reinforcement = 0

Both r_{eq} and the learning rates LRO and LRM determine the speed of learning. A higher r_{eq} can speed up the learning, especially for a problem like the MRCPSP where attempts only rarely result in improvements.
The settings of the 2 learning rates are dependent. A proper combination will be important for a good overall performance.

The viewpoint of a single agent is presented in Figure 4. Each agent has two learning devices. When the agent is visited for the first time, the algorithm will ask an agent to choose an order to visit its successors and a mode. For both choices the agent consults the corresponding learning automaton. These learning automata make a choice according to their probability vector (probability distribution). When all the agents have been visited at least once, the algorithm constructs a schedule. Using the information from this schedule, the reward system will update all the agents according to the reinforcement (reward) rules mentioned above (Equation 11 and 12). The agents forward the reinforcement signal to their learning automata devices. These learning automata will then update their probability vector using the (L_{R-I}) method.

4. EXPERIMENTAL RESULTS

In this section we evaluate the performance of the multi-agent learning algorithm. The algorithm has been implemented in Java Version 6 Update 11 and run on an Intel Core 2 Duo E8400 3.0GHz processor, 4GB RAM. To test the performance of the algorithm, we applied it to instances of the project scheduling problem library (PSPLIB) [14], which is available from the ftp server of the University of Kiel (http://129.187.106.231/psplib/).

First we present the experimental results for the multi-mode RCPSP in Section 4.1. In Section 4.2 we consider the single-mode version.

4.1 Multi-Mode

The PSPLIB library contains a number of MRCPSP datasets with a number of activities ranging from 10 to 30 (J10, J12, J14, J16, J18, J20 and J30). For all except the last dataset the optimal solutions are known. All the instances from these datasets have two renewable and two nonrenewable resources. Each dataset contains 640 instances, of which some are infeasible. We exclude the infeasible instances from the tests.

When testing the algorithm we found that the required number of iterations depends largely on the initial settings. For that reason we used the algorithm in the common multi-restart way. This involves restarting the algorithm a number of times on the same instance and taking the best solution over all the runs. In Table 1,2,3 and 4 we present the results of the multi-agent based algorithm for the J10 to J20 datasets from the above mentioned PSPLIB library, using the following parameters for all the tests: 0.01 for the order learning rate (LRO), 0.2 for the mode learning rate (LRM), $r_{eq} = 0.05$, 0% Pr_{ToDisp}, 5% $Pr_{RandToVisited}$ and different $Restarts \times Iterations$ combinations each with a total of 100,000 iterations. For these $Restarts \times Iterations$ combinations we used: $5 \times 20,000$ iterations (5 restarts with 20,000 schedule constructions each), $10 \times 10,000$ iterations

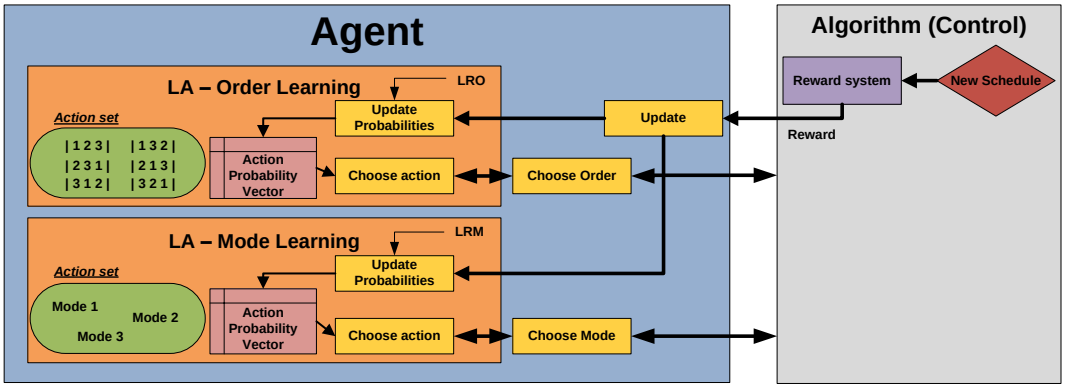


Figure 4: The single agent view.

and a tuned combination (see later). The results have been evaluated in terms of the average procentual deviation from optimum over all the instances or the relative error (RE), the maximum RE, the standard deviation over all RE, the % of optimal solutions found, and the average runtime in seconds. We compared our algorithm with an other agent based approach [9], a population learning algorithm (PLA) [8] and a simulated annealing algorithm [2]. For the population learning algorithm we took the results for 50,000 schedule constructions and for 2 PLA runs, which is similar to the total of 100,000 iterations of the agent based learning approach.

In Table 5 we present the results for the J30 dataset using $5 \times 20,000$ iterations and $5 \times 50,000$ iterations. For this dataset the optimal solutions have not been found by the research community. We therefore calculated the average procentual deviation from the best known solutions.

The learning rates LRO and LRM have been determined empirically by measuring the average performance of some learning rate combinations on several instances from the different datasets. In Figure 5 we present the average resulting makespan of different learning rate combinations for the J2054_2 instance. Here it seems that the $[0.01 - 0.2]$ and $[0.2 - 0.2]$ combinations perform best. Similar conclusions have been made when considering other instances. In any case, the LRM must be large enough (e.g. $LRM = 0.2$) to give good results in the limited interval of 20,000 iterations. This is probably due to the importance of choosing proper modes in the MRCPSP. We also added the learning rate combination $[0.0 - 0.0]$ which means that the agents do not learn, but select random actions. As we expected the method without learning performs the worst.

To determine the number of restarts together with the number of iterations per restart we did some experimental tests on the hardest instances for every dataset(i.e. instances for which our approach performed the worst in earlier tests). For every hard instance we performed 20 runs for some $Restarts \times Iterations$ combinations. These combinations all have a total of 100,000 iterations. We averaged the procentual difference with the optimum over the 20 runs. All tests have been executed using the default parameter values mentioned in the beginning of this Section. In Figure 6 we see the results for some hard J20 instances, which shows us that the $10 \times 10,000$ combination is the best performing for this dataset. We can draw similar conclusions from Figure

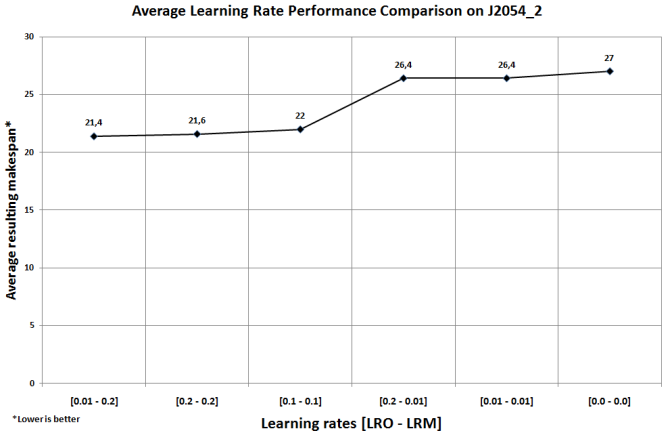


Figure 5: A comparison of different learning rate combinations for the J2054_2 instance.

7 and 8 for the J30 and J10 dataset. $5 \times 20,000$ seems the best combination for J30, while $50 \times 2,000$ appears to be the best for the J10 dataset. Using these best performing $Restarts \times Iterations$ combinations for every dataset, we obtained the 'Tuned' results from Figure 4. In general we can see that larger problem instances need more iterations, taking into account the fixed 100,000 iterations this automatically results in fewer restarts.

When considering these results for the MRCPSP we can conclude that the multi-agent approach performs very well when comparing it to the methods from the literature. We even reach 100% optimality for the J10 dataset when using the Tuned version of the algorithm.

4.2 Single-Mode

Since the MRCPSP is a more general definition than the RCPSP, the multi-agent learning approach is also suitable for solving the latter problem. In Table 6 we present the results for the J120 RCPSP dataset, which is the largest dataset for RCPSP in the PSPLIB library. The tests were carried out with the same parameters as in Section 4.1 but only $5 \times 5,000$ iterations. Since not all the optimal solutions are known for this dataset we calculate the average procentual deviation from the critical path length. We also provide the average procentual deviation from the best known solu-

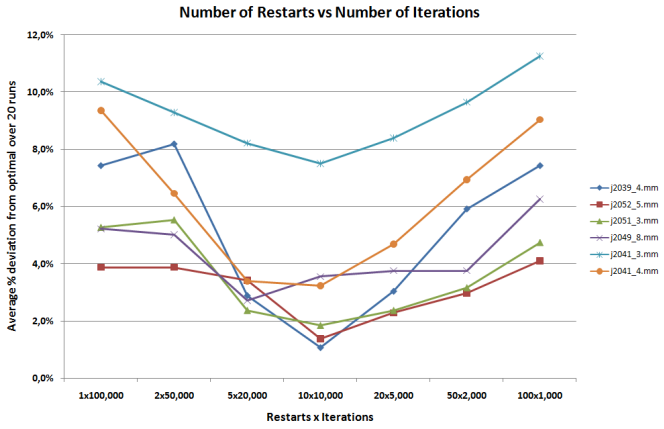


Figure 6: Number of restarts vs number of iterations for some hard J20 instances

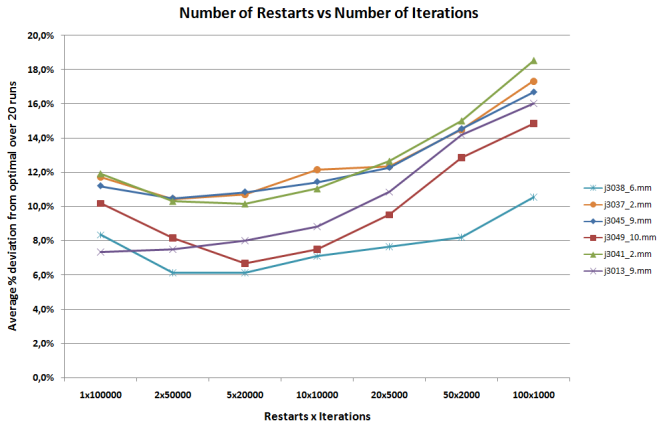


Figure 7: Number of restarts vs number of iterations for some hard J30 instances

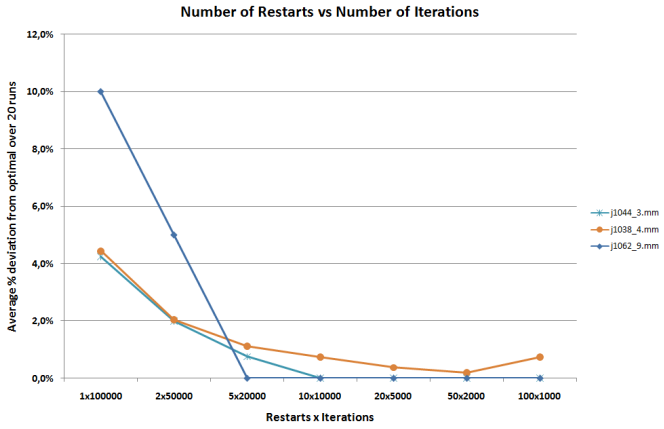


Figure 8: Number of restarts vs number of iterations for some hard J10 instances

tions.

When looking at these single-mode RCPSP results, which reveal average performance when comparing it to the best algorithms reported in the literature, we can conclude that the power of the approach is its coupling between learning the activity order and learning the modes.

Note that although our approach is distributed, it does not require mutual communication between the learning automata. The coupling of the LA happens through the common global reward signal. For both the RCPSP and MR-CPSP, specialized Genetic Algorithms (GA) are among the best performing algorithms in the literature. When we compare our results, with one of the very best GAs for the MR-CPSP [17], the results of the multi-agent learning approach have similar quality and even performs slightly better on some multi-mode datasets. However this comparison is not completely fair, because we did use more schedule constructions ($> 5,000$).

5. CONCLUSIONS

In this paper we have presented a novel approach for solving the multi-mode resource-constained project scheduling problem (MRCPSP) using agents. The agents make use of simple learning automata for learning both the activity orders and the mode assignments simultaneously.

Based on the results presented in this paper, we can conclude that the multi-agent approach performs very well when comparing it to the methods from the literature, especially to other agent-based and learning methods. In the future we will speed up the learning ($\pm 5,000$ iterations), by parameter tuning or incorporation of heuristic information (e.g. dispatcher strategy), so we can make a fair comparison with the most competitive algorithms, which are specialized Genetic Algorithms for the MRCPSP.

Instead of only testing the multi-agent approach on benchmarks, we expect that the presented approach is also capable of handling real practical problems.

The 'rough-and-ready' aspect of the experimental configuration coupled with the good results, strongly suggests a promising future for further research and the practical application of learning automata to several scheduling problems. Further on, this method can be applied to problems where one needs to find a permutation of elements which is restricted by precedence constraints, as in the Precedence Constraint Traveling Salesman Problem (PCTSP) or the Sequential Ordering Problem. Finally, we will also investigate how to relate the developed algorithm to the theoretical frameworks [18, 19] for interconnected LA learning devices.

6. REFERENCES

[1] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling projects subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[2] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149:268 – 281, 2003.

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models and methods. *EJOR*, 112:3–41, 1999.

Table 1: Experimental results MRCPSP

	J10	J12	J14	J16	J18	J20
Average deviation from optimal (RE) (%):						
P. Jedrzejowicz and E. Ratajczak (2007) [9]	0.72	0.73	0.79	0.81	0.95	1.80
P. Jedrzejowicz and E. Ratajczak (2006) [8]	0.36	0.50	0.62	0.75	0.75	**0.75**
K. Bouleimen and H. Lecocq (2003) [2]	0.21	0.19	0.92	1.43	1.85	2.10
Multi-Agent Learning Approach ($5 \times 20,000$ ***)***	**0.04**	**0.11**	**0.28**	**0.34**	**0.45**	0.81
Multi-Agent Learning Approach ($10 \times 10,000$ ***)***	**0.01**	**0.02**	**0.17**	**0.23**	**0.36**	**0.72**
Multi-Agent Learning Approach (Tuned)	**0.00**	**0.02**	**0.11**	**0.18**	**0.36**	**0.72**
Average runtime (s)	5	6.5	7.5	9	10	11.5

Table 2: Experimental results MRCPSP - $5 \times 20,000$

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.04**	**0.11**	**0.28**	**0.34**	**0.45**	0.81
Std. Dev. RE (%)	0.55	0.79	1.08	1.20	1.36	1.83
Max RE (%)	11.11	8.70	7.69	8.70	12.50	10.71
Optimal (%)	99.44	97.99	93.47	91.82	88.41	80.40

Table 3: Experimental results MRCPSP - $10 \times 10,000$

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.01**	**0.02**	**0.17**	**0.23**	**0.36**	**0.72**
Std. Dev. RE (%)	0.17	0.28	0.80	0.93	1.14	1.67
Max RE (%)	4	4.76	5	6.25	6	10
Optimal (%)	99.81	99.63	95.64	93.64	90.40	82.03

Table 4: Experimental results MRCPSP - Tuned

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.00**	**0.02**	**0.11**	**0.18**	**0.36**	**0.72**
Std. Dev. RE (%)	**0.00**	0.29	0.63	0.82	1.14	1.67
Max RE (%)	**0.00**	5.56	5.26	7.14	6	10
Optimal (%)	**100**	99.63	96.73	94.91	90.40	82.03

Table 5: Experimental results MRCPSP - J30

	Average deviation from best known solutions (%)	Max RE (%)	Average runtime (s)
$5 \times 20,000$ iterations	2.03	16.13	39
$5 \times 50,000$ iterations	1.10	11.90	157

Table 6: Experimental results RCPSP - J120

	Average deviation from critical path length (%)	Average deviation from best known solutions (%)	Average runtime (s)
$5 \times 5,000$ iterations	36.98	4.36	120

[4] S. Hartmann. Project scheduling with multiple modes: a genetic algorithm. *Annals of Operations Research*, 102:111–135, 1997.

[5] W. Herroelen and B. De Reyck. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *EJOR*, 119:538–556, 1999.

[6] W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-constrained project scheduling: a survey of recent developements. *Computers and Operations Research*, 25:297–302, 1998.

[7] J.Alcaraz and C. Maroto. A new genetic algorithm for the multi-mode resource-constrained project scheduling problem. page 4, 2002.

[8] P. Jedrzejowicz and E. Ratajczak. *Population Learning Algorithm for the Resource-Constrained Project Scheduling*, volume 92 of *International Series In Operations Research & Management Science*, chapter 11, pages 275 – 296. Springer US, 2006.

[9] P. Jedrzejowicz and E. Ratajczak-Ropel. Agent-based approach to solving the resource constrained project scheduling problem. 4431/2007(8th International Conference, ICANNGA 2007):480–487, 2007.

[10] M. D. G. Knotts. Agent-based project scheduling. *IIE Transactions*, 32:387–401, 2000.

[11] R. Kolisch. Project scheduling under resource constraints - efficient heuristics for several problem cases. *Physica-Verlag*, 1995.

[12] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project-scheduling problem: Classification and computational analysis. *Handbook on recent advances in project scheduling*, 1998.

[13] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006.

[14] R. Kolisch and A. Sprecher. Psplib - a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[15] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[16] M. Masao and C. Tseng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *EJOR*, 100:134–141, 1997.

[17] V. Van Peteghem and M. Vanhoucke. A genetic algorithm for the multi-mode resource-constrained project scheduling problem. Working paper, January 2008.

[18] P. Vrancx, K. Verbeeck, and A. Nowé. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics*, 38(4):976 – 981, August 2008.

[19] R. M. Wheeler and K. Narendra. Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control*, AC-31:519 – 526, 1986.

Local Optimal Solutions for DCOP: New Criteria, Bound, and Algorithm

Zhengyu Yin, Christopher Kiekintveld, Atul Kumar, and Milind Tambe
Computer Science Department
University of Southern California, Los Angeles, CA, 90089
{zhengyuy, kiekintv, atulk, tambe}@usc.edu

ABSTRACT

Distributed constraint optimization (DCOP) is a popular formalism for modeling cooperative multi-agent systems. In large-scale networks, finding a global optimum using complete algorithms is often impractical, which leads to the study on incomplete algorithms. Traditionally incomplete algorithms can only find locally optimal solution with no quality guarantees. Recent work on k-*size-optimality* has established bounds on solution quality, but size is not the only criteria for forming local optimization groups. In addition, there is only one algorithm for computing solutions for arbitrary k and it is quite inefficient. We introduce t-*distance-optimality*, which offers an alternative way to specify optimization groups. We establish bounds for this criteria that are often tighter than those for k-optimality. We then introduce an asynchronous local search algorithm for t-*distance-optimality*. We implement and evaluate the algorithm for both t and k optimality that offer significant improvements over KOPT – the only existing algorithm for k-*size-optimality*. Our experiment shows t-*distance-optimality* converges more quickly and to better solutions than k-*size-optimality* in scale-free graphs, but k-*size-optimality* has advantages for random graphs.

1. INTRODUCTION

In various cooperative multi-agent domains, agents have limited information and can directly communicate with only a subset of other agents. Typically, in these domains, the utility generated by an individual action of one agent depends only on the actions of a set of nearby agents. Distributed constraint optimization (DCOP) is a popular formalism for modeling such cooperative multi-agent systems in which agents work together to optimize a global objective. The objective function can be decomposed into constraints with associated utility matrixes across local subsets. There are lots of applications of DCOP including multi-agent plan coordination [5], sensor networks [15], allocation of resources in peer-to-peer networks [6], and meeting scheduling [13].

Various complete algorithms have been developed for finding globally optimal solution to DCOPs, e.g. ADOPT [10], OptAPO [9], DPOP [13], and NCBB [4]. However, DCOP is NP-hard [10]. It is hard for complete algorithms to scale up because the computation burden might increase exponentially with the increasing number of variables. Therefore, researchers have been studying incomplete algorithms, in which agents normally form local groups with small size and optimize within these groups. Incomplete algorithm have better scalability and robustness in dynamic environments. Existing incomplete algorithms include MGM/DBA [12, 15] and DSA [7]. Recent studies with k-*size-optimality* by Pearce et al. [11] have begun to provide theoretical quality guarantees of locally optimal solutions with certain properties. k-*size-optimality* is characterized by the size of local groups, i.e. the minimum number of variables which must change their values to improve the overall solution quality.

Unfortunately, k-*size-optimality* suffers from several shortcomings. First, our theoretical analysis shows the quality lower bound for k-*size-optimality* is inversely related to the graph density. The worst case is complete graphs, where the lower bound for k-*size-optimality* decreases to $\frac{k-1}{2n-k-1}$ [11]. As n increases, this bound gets unacceptably low for some domains. Second, k-*size-optimality* considers all types of k-size groups, among which some may have nodes that are far apart from each other. In such groups, coordination to perform a joint move can be expensive, as messages have to be delivered to some nodes far away from the center node. Third, the number of possible k-size groups in the graph grows combinatorially with increasing values of k. Basically, any combination of k connected nodes can form a k-size group. The complexity of enumerating and optimizing all k-size groups makes it difficult to design efficient algorithms for k-*size-optimality* with $k \geq 3$. This explains why most local search algorithms have limited group size to 1 or 2. "KOPT" [8], recently introduced by Katagishi and Pearce, is the only known algorithm for k-*size-optimality* that supports arbitrary values of k. However it suffers from significant messaging overhead, particularly in highly dense graphs.

In this paper, we introduce a new locally optimal criteria – t-*distance-optimality* which has fixed number of local optimization groups defined by graph distance. We provide formal quality bounds for t-*distance-optimal* solutions with and without prior knowledge about graph structure, and show these bounds can yield significantly better guarantees than comparable k-*size-optimal* solutions. Furthermore, we present an asynchronous local search algorithm for t-*distance-optimality* based on standard lock/commit protocol, and show its significant improvements over KOPT in various metrics. Finally we conducted comprehensive experiments to evaluate tradeoffs between algorithms for t-*distance-optimality* and k-*size-optimality* considering a variety of metrics. Our experiments show t-*distance-optimality* converges more quickly and to better solutions than k-*size-optimality* in scale-free graphs, but k-*size-optimality* has advantages in random graphs.

2. BACKGROUND

Cite as: Local Optimal Solutions for DCOP: New Criteria, Bound, and Algorithm, Z. Yin, C. Kiekintveld, A. Kumar, and M. Tambe, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

2.1 DCOP Definition

A finite *distributed constraint optimization* (DCOP) problem comprises sets of variables $V := \{v_1, \ldots, v_n\}$ and constraints $C := \{c_1, \ldots, c_m\}$. Variables have finite domains and are each controlled by a decision-making agent (for convenience we assume one variable per agent). A joint assignment for all variables is given by $A := \{a_1, \ldots, a_n\}$. We follow the convention in the literature and consider only binary constraints. For some pair of variables (v_i, v_j), a constraint c defines a real-valued reward for all possible joint assignments, $c(a_i, a_j)$. The reward function $R(\cdot)$ of an assignment A is defined as the sum of rewards, $R(A) = \sum_{c \in C} c(a_i, a_j)$. The agents' objective is to find an assignment A^* that maximizes the reward function, i.e. $A^* = \arg\max_A R(A)$. The *constraint graph* has a node for each variable and an edge for each constraint. In the sequel, we use the terms node, variable and agent interchangeably. Agents initially know only their own constraints, and can communicate only with neighbors in the constraint graph.

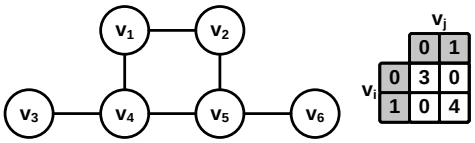


Figure 1: An example DCOP with six binary variables. Each constraint has the same reward table.

DCOP shown in figure 1 contains 6 variables, and 6 constraints with the same reward matrix. The optimal assignment of this problem is $A = \{1, 1, 1, 1, 1, 1\}$ with a reward $R(A) = 24$.

2.2 k-Size Optimality

Pearce et. al. recently introduced *k-size-optimality* (as "k-optimality" in the original work) as a local optimality criteria that offers theoretical guarantees on solution quality. In their approach, agents form groups of one or more agents until no group of k or fewer agents can possibly improve the DCOP solution. This type of local optimum is defined as *k-size-optimal*. Let A be a DCOP assignment and $A(i)$ be the value of v_i in A.

DEFINITION 1. *The deviation group $D(A, A')$ is defined as the set of nodes with a different assignment in A and A', i.e. $D(A, A') = \{v_i | A(i) \neq A'(i)\}$.*

DEFINITION 2. *A DCOP assignment A is* k-size optimal *if $R(A) \geq R(A')$ for all A' for which $|D(A, A')| \leq k$.*

In Figure 1, the assignment $A = \{0, 0, 0, 0, 0, 0\}$ with reward $R(A) = 18$ is a k-size optimal solution for $k = 1, 2, 3, 4$, but not for $k = 5, 6$. It is 1-size optimal because the reward is reduced if any single variable changes assignment, and by carefully examining all possibilities it can also be proved to be $2, 3, 4$-size optimal. However, it is not 5-size-optimal because if we change the values of v_1, v_2, v_3, v_4, and v_5 from 0 to 1, the solution reward is improved to 20. For any binary DCOP with n variables, a k-size optimal solution A has quality $R(A) \geq \frac{k-1}{2n-k-1} R(A^*)$, where A^* is the globally optimal solution [11]. This bound is independent on the graph structure and reward structure, but tighter bounds are possible given additional information about the problem [3]. Many incomplete DCOP algorithms including MGM and DSA yield 1-size optimal solutions. General *k-size-optimality* offers a spectrum of solutions with stronger guarantees in exchange for greater computation [11].

3. T-DISTANCE OPTIMALITY

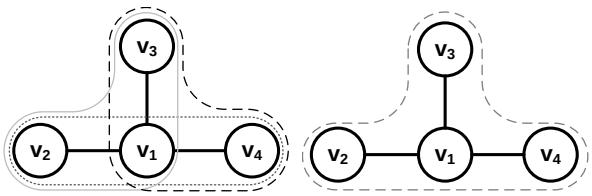
We introduce a novel local optimality criteria, *t-distance-optimality*, that defines local optimization groups based on graph distance. We begin with a formal definition, and discuss the relationship between *k-size-optimality* and *t-distance-optimality*. Then, we give a general lower bound on solution quality of *t-distance-optimality* regardless of graph structure. Finally, we present graph-specific bounds on sets of graphs with varying properties.

For a pair of variables u and v, let $T(u, v)$ be the shortest distance between them in the constraint graph.

DEFINITION 3. *Denote by $\Omega_t(v)$ the group of variables that can be reached from v within t hops, i.e. $\Omega_t(v) = \{u | T(u, v) \leq t\}$.*

DEFINITION 4. *A DCOP assignment A is* t-distance optimal *if $R(A) \geq R(A')$ for all A', where $D(A, A') \subseteq \Omega_t(v)$ for some $v \in V$.*

There are at most n distinct t-distance groups centered on n variables. Some groups may be redundant when they are subsumed by or equivalent to others. For example, in a complete graph, only one group is not redundant because all n 1-distance groups comprise the full set of variables and are thus identical. Furthermore, consider the example shown in Figure 1. There are no two identical 1-distance groups, however $\Omega_1(v_3) = \{v_3, v_4\}$ is subsumed by $\Omega_1(v_4) = \{v_3, v_4, v_1, v_5\}$, leading to redundance of $\Omega_1(v_3)$. To better understand the difference between t-groups and k-groups, we consider another example shown in Figure 2. Figure 2(a) shows all three 3-size groups in the graph: $\{v_1, v_2, v_3\}$, $\{v_1, v_3, v_4\}$, and $\{v_1, v_2, v_4\}$. Figure 2(b) shows the only non-redundant 1-distance group: $\{v_1, v_2, v_3, v_4\}$.



(a) 3-size groups (b) 1-distance groups

Figure 2: k-size groups and t-distance groups

Consider, again, the example in Figure 1, where we show assignment $A = \{0, 0, 0, 0, 0, 0\}$ is 0-distance optimal with quality $R(A) = 18$. In 0-distance optimality, each group contains exactly one variable, which is equivalent to 1-size optimality. Changing the value of any single variable from 0 to 1 will reduce the quality. This is because flipping the value of either v_3 or v_6 leads to reward 15, flipping the value of either v_1 or v_2 leads to reward 12, and flipping the value of either v_4 or v_5 leads to reward 9. Since none of these groups has an incentive to deviate, A is 0-distance optimal. Similarly we can test whether A is 1-distance optimal by checking all 1-distance groups. In this example, there are four non-redundant 1-distance groups: $\Omega_1(v_4) = \{v_1, v_3, v_4, v_5\}$, $\Omega_1(v_1) = \{v_1, v_2, v_4\}$, $\Omega_1(v_2) = \{v_1, v_2, v_5\}$, and $\Omega_1(v_5) = \{v_2, v_4, v_5, v_6\}$. After enumerating all possible deviations in these groups, we find no group can improve solution quality by local optimization. Therefore A is also 1-distance optimal. However, it is not 2-distance optimal because $\{1, 1, 1, 1, 1, 1\}$ is a better assignment whose deviation group is fully comprised by $\Omega_2(v_4)$. In this example, 2-distance optimality already guarantees global optimality because $\Omega_2(v_4)$ contains exactly all variables in the network.

3.1 Comparing t and k Optimality

Both *t-distance-optimality* and *k-size-optimality* are criteria for local optimality, but there is a key distinction between them. In *k-size-optimality*, the size of local optimization group is fixed, but the number of possible k-size groups may be very large, especially in dense graphs. In contrast, in *t-distance-optimality*, the number of optimization groups is fixed, but the size of t-distance groups can be very large, particularly in dense graphs. For example, in a complete graph with n variables, there are $\binom{n}{3}$ distinct 3-size groups where each has a fixed size 3. However, there is only one unique 1-distance group comprising all n variables. One of our primary contributions in this paper is to empirically test the implications of this tradeoff for local search methods.

To further understand the connection between k-optimality and t-optimality, we examine some special cases. The first observation is that 1-size optimality is equivalent to 0-distance optimality. In both criteria, one single node forms the local optimization group. Furthermore, in ring graphs, *t-distance-optimality* is also equivalent to *k-size-optimality* for $k = 2t + 1$. For example, 3-size optimality is exactly the same as 1-distance optimality in a ring graph, because every section of 3 connected nodes is an optimization group for both $k = 3$ and $t = 1$ and no additional groups exist for either case.

There are several reasons to believe *t-distance-optimality* potentially offers benefits over *k-size-optimality* as a criterion for distributed local search algorithms. First of all, a t-distance optimal solution always guarantees $(2t + 1)$-size optimality, since every $(2t + 1)$-size group must be contained in some t-distance group. But the reverse is not true; there are $(2t + 1)$-size optimal solutions that are not t-distance optimal. For example, in a complete graph, t-distance optimal solution always guarantees global optimality for $t \geq 1$, while a k-size optimum can possibly reach the lower bound given in Section 2.2 which is known to be tight for complete graphs. While complete graphs are a somewhat artificial example, we might expect to find similar advantages for t-distance optimality when there are hub nodes with many connections or subgraphs that are densely connected. Second, *t-distance-optimality* naturally captures graph locality which might help improving efficiency of local search algorithms particularly in distributed environment where communication delay is the dominating cost. Also, algorithms for *t-distance-optimality* may reduce privacy loss as private information of an agent can only be obtained by those within a fixed distance.

3.2 General Lower Bound

We derive a general lower bound on solution quality for *t-distance-optimality*, regardless of the graph structure.

PROPOSITION 1. *Consider a DCOP with n variables, minimum constraint arity m, where all constraint rewards are non-negative, and A^* is the globally optimal solution, then, for any t-*distance optimal *assignment A_{topt}, where $t > 0$ and $m+t-1 \leq n$, we have $R(A_{topt}) \geq \frac{m+t-1}{n} R(A^*)$.*

PROOF. Let $R_c(A)$ denote the reward on constraint c for any assignment A. For any set of constraints S, let $R_S(A) = \sum_{c \in S} R_c(A)$. Let $\sigma(c)$ be the set of variables in constraint c, and $\pi(W)$ be the set of constraints across a subset of variables $W \subseteq V$ ($c \in \pi(W)$ iff $\sigma(c) \subseteq (W)$).

Let $A'(v)$ be an assignment derived from A_{topt} by changing all assignments in $\Omega_t(v)$ to their corresponding values in A^*. Since A_{topt} is a *t-distance optimal* assignment, $R(A_{topt}) \geq R(A'(v))$. Because all constraint values in the DCOP are non-negative, $R(A'(v)) \geq R_{\pi(\Omega_t(v))}(A'(v))$. Furthermore $A'(v)$ has the identical assignment as A^* over $\Omega_t(v)$, so

$$R(A_{topt}) \geq R_{\pi(\Omega_t(v))}(A'(v)) = R_{\pi(\Omega_t(v))}(A^*)$$

Summing over all t-groups, we have:

$$nR(A_{topt}) \geq \sum_{i=1}^{n} R_{\pi(\Omega_t(v_i))}(A^*) \quad (1)$$

Now we count the contribution of each constraint c to the rhs. Because c has an arity of at least m, $|\sigma(c)| \geq m$. Pick an arbitrary variable v in $\sigma(c)$, if the t-group $\Omega_t(v)$ is identical to V, i.e. comprising all variables in the graph, then $R(A_{topt}) = R(A^*)$. Otherwise, there exists a $v_j \in V$ such that $T(v, v_j) > t$. Write the first $t+1$ variables on the shortest path from v to v_j as $v, v_1, \ldots, v_t$. $T(v, v_i) = i$, which implies that for $i > 1$, $v_i \notin \sigma(c)$.

Consider two cases. First, if $v_1 \in \sigma(c)$, c appears in $\pi(\Omega_t(v'))$ for all $v' \in \sigma(c) \cup \{v_2, v_3, ..., v_t\}$. Certainly c appears $\pi(\Omega_t(v'))$ for $v' \in \sigma(c)$ as $t \geq 1$. c also appears $\pi(\Omega_t(v'))$ for $v' \in \{v_2, v_3, ..., v_t\}$ because for any variable v'' in $\sigma(c)$ and any $2 \leq i \leq t, T(v'', v_i) \leq T(v'', v_1) + T(v_1, v_i) = 1 + i - 1 \leq t$. Therefore in the rhs of inequality 1, c is counted at least $|\sigma(c)| + t - 1 \geq m + t - 1$ times.

Now consider the other case where $v_1 \notin \sigma(c)$. c appears in $\pi(\Omega_t(v'))$ for all $v' \in \sigma(c) \cup \{v_1, v_2, ..., v_{t-1}\}$. c appears $\pi(\Omega_t(v'))$ for $v' \in \{v_1, v_2, ..., v_{t-1}\}$ because for any v'' in $\sigma(c)$ and any $1 \leq i \leq t - 1, T(v'', v_i) \leq T(v'', v) + T(v, v_i) = 1 + i \leq t$. Therefore, c will be also counted at least $|\sigma(c)| + t - 1 \geq m + t - 1$ times in this case. Since c is counted at least $m + t - 1$ times in the rhs of inequality 1 in both cases we have,

$$R(A_{topt}) \geq \frac{\sum_c (m + t - 1) R_c(A^*)}{n} = \frac{(m + t - 1)}{n} R(A^*)$$

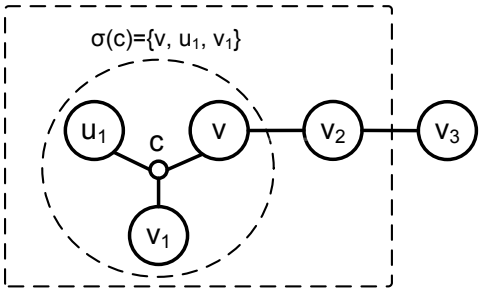
□

To demonstrate the two cases in the proof above, we consider the example shown in Figure 3, where $|c| = 3$ and $t = 2$. In this example, we show constraint c appears in $|c| + t - 1 = 4$ distinct 2-distance groups. Figure 3(a) demonstrates the situation where $v_1 \in \sigma(c)$. In this case, constraint c appears in four groups centered on v, u_1, v_1, and v_2 respectively. Figure 3(b) shows the other situation where $v_1 \notin \sigma(c)$. In this case, constraint c appears in four groups centered on v, u_1, u_2, and v_1 respectively. In both figures, variables inside the dashed circle represent $\sigma(c)$ and variables inside the dashed box represent those whose t-groups have constraint c. As we can see in both cases, c appears in at least 4 different 2-distance groups.

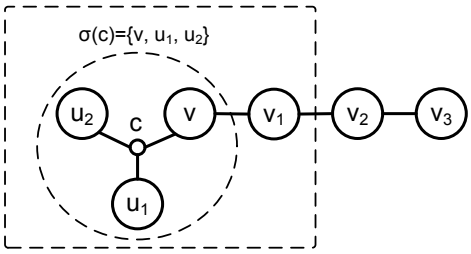
PROPOSITION 2. *For binary constraint DCOP with n variables, the lower bound for 1-distance optimality in proposition 1 is tight.*

PROOF. Consider a complete bipartite graph with $2h$ variables (figure 4 shows an example of $h = 3$). Let $S_1 = \{v_{11}, v_{12}, \ldots, v_{1n}\}$ and $S_2 = \{v_{21}, v_{22}, \ldots, v_{2n}\}$. For any $1 \leq i, j \leq n$, there is a constraint between v_{1i} and v_{2j}. Variables can take a value of either 0 or 1. All constraints have the same reward matrix as shown in figure 4. The global optimum is $\{1, 1, \ldots, 1\}$ with quality h^3. Proposition 1 gives the lower bound for 1-distance optimum of $\frac{2}{2h} h^3 = h^2$. We claim that assignment $\{0, 0, \ldots, 0\}$ is an 1-distance optimum with quality h^2. Consider only variable v_{11}, w.l.o.g. due to symmetry. $\Omega_1(v_{11})$ contains all h variables in S_2 and none in S_1.

i. Suppose the value of v_{11} remains 0, and $1 \leq b \leq n$ variables in S_2 change to 1, then the total quality will decrease to $h(h - b)$.



(a) In the first case, constraint c appears in groups centered on v, u_1, v_1, and v_2



(b) In the second case, constraint c appears in groups centered on v, u_1, u_2, and v_1

Figure 3: Example showing the two cases

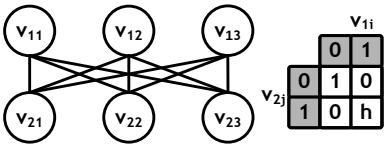


Figure 4: Example showing bound tightness for $t = 1$

ii. Suppose the value of v_{11} is changed to 1, and $0 \leq b \leq n$ variables in S_2 change to 1, then the total quality will be to $bh + (h-1)(h-b) = h^2 - h + b \leq h^2$.

In either case, the solution quality can't be improved. Therefore $\{0, 0, \ldots, 0\}$ is 1-distance optimal. □

3.3 Graph-Specific Lower Bound

In previous work on *k-size-optimality*, linear fractional programming (LFP) was used to find tighter bounds for specific graphs [11]. We use a similar method for t-distance optimality. One LFP variable $R_c(A_{topt})$ represents the reward on c in the t-distance optimal solution, and a second $R_c(A^*)$ represents the reward in the optimal solution. By definition $R(A_{topt}) \geq R(A')$ for all A', where the deviation group between A_{topt} and A' is comprised in some t-group. Let Θ be the set of assignments such that $A' \in \Theta$ iff $D(A_{topt}, A') \subseteq \Omega_t(v)$ for some $v \in V$ and variables in $D(A_{topt}, A')$ take the same value as in A^*. The objective is to minimize $\frac{R(A_{topt})}{R(A^*)}$ such that $\forall A' \in \Theta, R(A_{topt}) - R(A') \geq 0$. Note that $R(A_{topt})$ and $R(A^*)$ can be expressed as $\sum_c R_c(A_{topt})$ and $\sum_c R_c(A^*)$. $R(A')$ can also be represented as the sum over all constraints, $R(A') = \sum_{c \in C} R_c(A')$. So far we can write down the LFP:

$$
\begin{aligned}
Min \quad & \frac{R(A_{topt})}{R(A^*)} \\
s.t. \quad & R(A_{topt}) = \sum_{c \in C} R_c(A_{topt}) \\
s.t. \quad & R(A^*) = \sum_{c \in C} R_c(A^*) \\
s.t. \quad & R(A') = \sum_{c \in C} R_c(A'), \forall A' \in \Theta \\
s.t. \quad & R(A_{topt}) \geq R(A'), \forall A' \in \Theta
\end{aligned}
$$

In the LFP above we still need to represent $R_c(A')$. Consider any constraint c, suppose it has two variables v_i and v_j. Denote by $A(i)$ the value of v_i in assignment A. Then,

i. $R_c(A') = R_c(A_{topt})$, if $A'(i) = A_{topt}(i) \wedge A'(j) = A_{topt}(j)$.

ii. $R_c(A') = R_c(A^*)$, if $A'(i) = A^*(i) \wedge A'(j) = A^*(j)$.

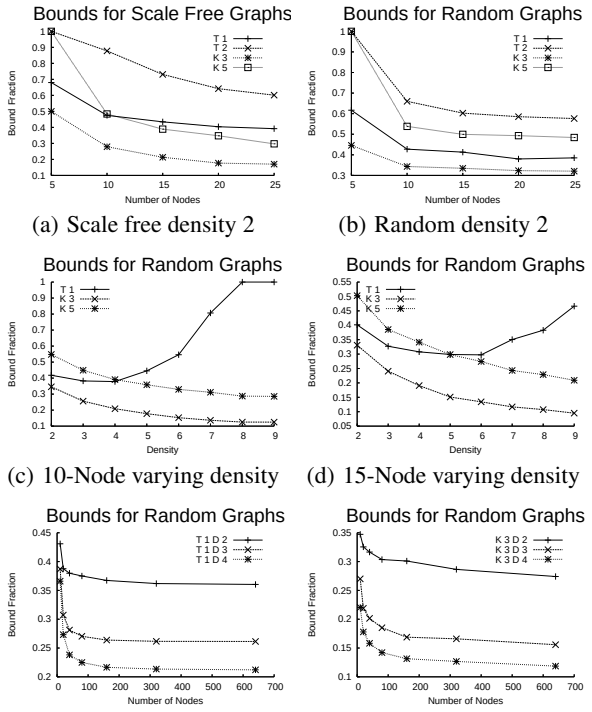
iii. $R_c(A') = 0$, otherwise.

Figure 5 shows the average graph-specific bounds over 30 samples. (see Section 5 for details on graph generation). In Figure 5(a) and Figure 5(b), we see that *t-distance-optimality* provides much stronger lower bounds on average than comparable *k-size-optimality* on both scale free and random graphs. We note $t = 2$ provides a substantial improvement over $t = 1$. Comparing lower bounds on different types of graphs, we also note *t-distance-optimality* offers more benefits on scale free graphs than on random graphs. The lower bound for $t = 1$ is generally better than that for $k = 5$ on scale free graphs while $k = 5$ constantly outperforms $t = 1$ on random graphs. The differences shown are statistically significant (for example, the p-value for a comparison of $t = 1$ and $k = 3$ on 25-node scale free graphs is 4.58×10^{-24}). We also did tests on graphs with varying density. Figure 5(c) and Figure 5(d) show the results for $t = 1$, $k = 3$, and $k = 5$ on 10-node and 15-node random graphs with varying density respectively. We note that k-size optimal bounds tend to degrade more quickly with increasing density, which t-distance optimality is more stable and eventually improves for very high densities. Finally, we tested bounds on large graphs. Figure 5(e) and Figure 5(f) show the bounds for $t = 1$ and $k = 3$ on random graphs with varying graph size. We can see both *t-distance-optimality* and *k-size-optimality* bounds degrade quickly at relatively small graph size. However, after graph size reaching 100, both bounds begin to stabilize and eventually converge to some fixed value. We can see $t = 1$ on average guarantees 21.2% of the global optimum on density 4 graphs with 640 nodes while $k = 3$ guarantees 11.9%.

4. ASYNCHRONOUS ALGORITHM

In this section, we introduce our novel asynchronous algorithm for DCOP based on *t-distance-optimality*. It is a variant of distributed local search which improves quality monotonically over time from a random initial assignment. It is fully distributed and uses asynchronous methods for both computation and coordination.

We give an overview of the algorithm before discussing key stages in more detail. First, agents broadcast local information about the graph structure and use this to determine group members. During the main phase, all groups compute new optimal assignments in parallel, assuming nodes outside of the group maintain unchanged. If an improvement is found, the group leader attempts to implement the new assignment by sending out requests. This can cause



(a) Scale free density 2 (b) Random density 2

(c) 10-Node varying density (d) 15-Node varying density

(e) Bounds for $t = 1$ (f) Bounds for $k = 3$

Figure 5: Graph-specific bounds comparison

conflicts among overlapping groups, which is resolved by an asynchronous locking and commitment protocol in our approach.

4.1 Initialization

An agent begins by picking a random value from its domain as its initial assignment. It then composes a message containing all its constraints and broadcasts the message to a distance of t hops. After that, it broadcasts its initial value to a distance of $t + 1$ hops in a separate message. Here we assume each agent has an unique ID (e.g. a MAC address) which can be included in the message to identify the sender.

In the initial stage, the center node of each group will receive full constraint information and values of nodes inside the group. Since the initial values are broadcast an additional hop, the center node will also receive the values of all fringe nodes of the group. Fringe nodes of a group are those outside the group but directly linked to some node inside the group. Local optimization of a group is performed under the assumption that all the fringe nodes remain static.

Every agent in the network will automatically be the leader of a t-distance group centered on it. However, some groups may be subsumed by or equivalent to others, leading to redundant computations. We remove some redundant groups using a simple optimization. Each agent computes a shortest distance matrix within its local view ($t + 1$ hops). Its t-group is redundant if

i. there is an agent whose shortest distance to every other agent is strictly less than $t + 1$;

ii. there is an agent with a lower id whose shortest distance to every other agent is less than or equal to $t + 1$.

4.2 Computation for Local Optimum

In the previous stage, the leader has gathered all the information required for computing the local optimal solution. During the main phase of the algorithm new optimal assignments for each group are computed in parallel. In principle, any complete DCOP solver can be adapted to this purpose. We implement a centralized approach motivated by OptAPO [9] for our experiments.

Our implementation uses a variable elimination algorithm comparable to a centralized version of DPOP [13] if the average density of the local perspective is less than $\frac{n}{2}$ and a branch-and-bound solver otherwise. Variable elimination methods like DPOP are exponentially complex with respect to the width of the pseudo tree. Therefore, the centralized DPOP solver can be very efficient in low density graphs. However, a branch-and-bound solver has advantage in dense graphs thanks to the pruning on solution space based on reward structure. To perform this computation, the leader needs to know the current assignments of the fringe nodes. Each time a node commits a change and switches to a new assignment, it broadcasts the new value to a distance of $t + 1$ hops to ensure that all leaders have the required information. Once the leader receives new assignment information, it immediately gives up the ongoing lock attempt (if there is one), unlocks all group members, and recomputes a new optimal assignment.

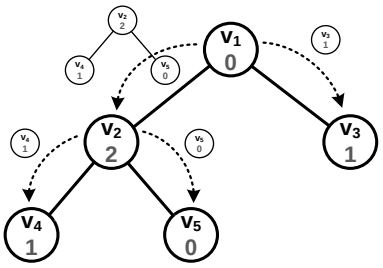
4.3 Asynchronous Assignment Implementation

When the leader finds the local optimum of the group that can yield improved solution quality, it will attempt to implement the new assignment. While implementing changes in a single group guarantees monotonically increasing quality, multiple overlapping groups implementing assignments simultaneously might lead to degradations. Conflicts happen when some agent receives exclusive value changing requests from multiple group leaders. Existing incomplete DCOP solvers, including KOPT [8], typically requires synchronization to resolve such conflicts. However, in real problems, synchronization over the whole network is not trivial to achieve in the first place. Furthermore, synchronous algorithms can be fragile in dynamic environments where message loss, node failures, and other unpredictable events happen frequently. For example, when a node loses connection to the network, synchronous algorithm completely stops working due to the failure of synchronization. In our algorithm, we use an asynchronous protocol, which is generally more robust to various types of noise in real applications. In asynchronous algorithms, when a single node failure happens, only a small set of nodes are affected and the remaining part of the network still works as normal.

The protocol is based on a standard lock/commit pattern. The leader sends lock requests to all group members and fringe nodes, which accept the request unless they have already locked on a different assignment (multiple locks for the same assignment are acceptable). If all nodes accept, the leader sends a commit message and the assignment is implemented. Otherwise, the leader unlocks all nodes and backs off to prevent deadlock.

It is trivial for the leader to lock the group when all lock requests are sent to its direct neighbors. However, we need a protocol to forward lock requests in large groups where the leader cannot directly communicate with all group and fringe nodes. A simple flooding approach based on broadcasting is inefficient, especially in dense graphs. Our algorithm fully utilizes the graph information procured in the initial stage to reduce number of messages. The leader first performs a breadth-first search and builds a BFS tree rooted on itself, in which a tree node represents an agent. A tree node is assigned an **(ID, Val)** pair, indicating agent **ID** is locked on **Val**. Each neighbor of the leader corresponds to a child of the root in the BFS tree. The leader sends the corresponding subtree to every neighbor

as a lock request. An agent that receives a lock message extracts the lock value from the root, and forwards the subtrees to its children. Figure 6 is a demonstration for this protocol.



6: Example demonstrating lock message forwarding protocol. Each node represents a variable v_i. The gray number in the node indicates the value to be locked on. Each arrow associated with a subtree that represents a lock message.

If a previous lock attempt fails, the leader will wait for a random interval between 0 and the maximum interval I_m before sending lock requests again. Failure of each lock attempt doubles I_m until reaching a threshold (256 time units). To reduce conflicts in the beginning, every leader waits for a random number of time units between 0 and 32 before the first lock attempt. When a node commits to an assignment, it broadcasts the new assignment to a distance of $t + 1$ hops and is able to accept new lock requests. Leaders receiving new assignment information unlock nodes if necessary and recompute a new optimal assignment. Conflicts have a substantial cost, so we implemented several techniques to improve locking performance.

Subset Locking (SL): If a new assignment does not change the assignment for every node, it is not necessary to lock all group and fringe nodes. Subset locking only requests locks from nodes changing assignment and their neighbors, rather than the full group, reducing the chance for conflicts.

Partial Synchronization (PS): A benefit of synchronized conflict resolution is that nodes can implement heuristics to commit to groups with a high gain for changing assignment. We approximate this in the asynchronous setting by waiting for a fixed time period to pool lock requests at each node. Once the timer expires, a node selects the lock message with the largest gain to accept (group gain is included in the lock message when using PS).

5. EXPERIMENTAL EVALUATION

In addition to the algorithm for *t-distance-optimality* described in Section 4, we implement a comparable algorithm for *k-size-optimality* using the same asynchronous coordination protocol. We explain some non-trivial details. First, in the initialization stage, constraints are broadcast to a distance of $\lfloor \frac{k}{2} \rfloor$ instead of t hops. And after changing value, agents broadcast the new value to a distance of $\lfloor \frac{k}{2} \rfloor + 1$ instead of $t + 1$ hops. Second, each agent can be the leader for multiple k-size groups. Each k-size group selects the most centralized agent (using ID as secondary comparator) to be the leader. Third, each agent computes the locally optimal solution for each k-size group it leads and locks the one that yields the maximum gain.

We test our algorithms in simulation, using the DAJ toolkit [14] which provides low-level support for simulating distributed algorithms in Java. This toolkit provides a simple programming interface that allows to develop distributed algorithms based on a message passing model. The primary performance metric we use is solution quality per unit of global time. Global time unit is motivated by the metric of cycles commonly used in evaluating synchronous DCOP algorithms. We made some small modifications to the DAJ toolkit so that in a single global time unit, each node can process all messages in the incoming queue, and send as many messages as desired to neighbors. Messages are delivered on the subsequent time steps, e.g. a message sent at time x will be received at time $x + 1$.

We generate DCOPs for three main classes of constraint graphs:

i. $G(n, M)$ random graphs [2]. In this model, the graph is generated by randomly adding M edges. Each is picked out of $\binom{n}{2}$ possible choices with equal probability. However, by excluding disconnected graphs to guarantee connectivity, our random graphs may be slightly different from the original model.

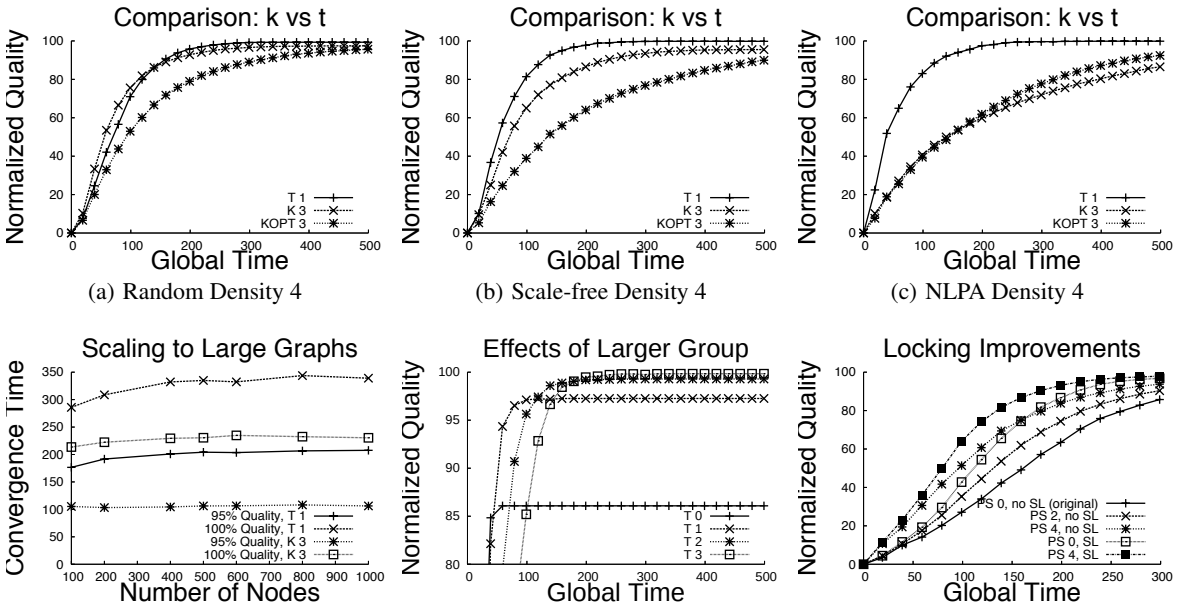
ii. Barabasi-Albert (BA) scale-free graphs [1]. This is one simple algorithm for generating random scale-free graphs using a linear preferential attachment mechanism. In this model, graphs begin with a complete graph of m_0 nodes, where m_0 is a small number but at least 2. New nodes are added to the network one at a time. Each new node is connected to $m \leq m_0$ of the existing nodes with a biased probability proportional to the number of links the existing node already has. The degree distribution resulting from the BA model is scale-free, in particular, it is a power law of the form $P(k) \sim k^{-3}$.

iii. Non-linear preferential attachment (NLPA) graphs based on the BA model, but with a stronger bias towards larger numbers of nodes with few connections. This model is based on the BA model, but instead of adding a new node to existing nodes with probability linear to their degree, we alter the probability to be non-linear to the degree, in particular, degree$^{1.7}$. As a result, heavily linked nodes ("hubs") tend to quickly accumulate even more links than those in scale-free graphs, while nodes with only a few links are less likely to be chosen as the destination for a new link.

All variables have domain size 10 and constraint rewards are all randomly drawn from the uniform distribution of integers in the range $[0 \ldots 10000]$.

Our first experiment compares k-size and t-distance optimality on random, scale free, and NLPA graphs. Each graph has 100 variables. Both k-size and t-distance algorithms use subset locking and partial synchronization. PS window size is selected after testing several possible settings. "KOPT" introduced by Katagishi and Pearce [8] is included as a benchmark. We show solution quality at each global time, averaged over 50 sample graphs. Each algorithm starts from the same random assignment. Quality is normalized by subtracting the value of the initial random assignment and dividing by the maximum value found by any algorithm for each problem instance. We do not include error bars so as not to clutter the graphs, but the primary comparisons described are all highly significant based on paired t-tests.

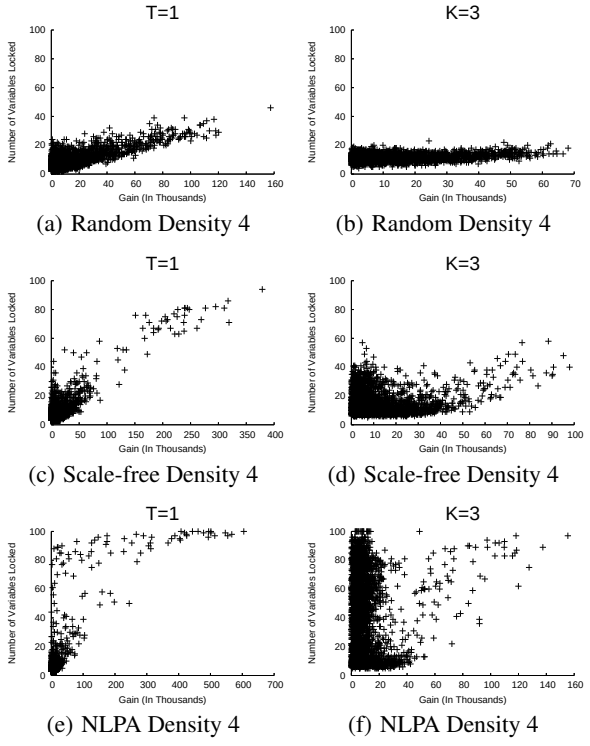
Results of the first experiment are shown in Figures 7(a), 7(b), and 7(c). Settings with $k = 2t + 1$ are comparable and equivalent for some classes of graphs (Section 3.1); we use settings of $k = 3$ and $t = 1$ to examine the tradeoff in local optimization group type. Our algorithms for both t-opt and k-opt clearly outperform KOPT on both random and scale-free graphs. Our *k-size-optimality* algorithm is similar to KOPT on NLPA, but *t-distance-optimality* strongly outperforms both other algorithms in this case. By time 100, its improvement in solution quality is double that of the other algorithms. In scale-free graphs *t-distance-optimality* is also a clear winner in both convergence speed and final quality. For random graphs, *k-size-optimality* has advantages in convergence speed, but converges to a lower final quality.



(a) Random Density 4 (b) Scale-free Density 4 (c) NLPA Density 4

(d) Random Density 4 (e) Random Density 2 (f) Random Density 4

Figure 7: Results of experiments (1) comparing k-size and t-distance optimality, (2) examining scalability to large graphs, (3) exploring the effects of increasing t, and (4) showing the benefits of locking improvements.



(a) Random Density 4 (b) Random Density 4

(c) Scale-free Density 4 (d) Scale-free Density 4

(e) NLPA Density 4 (f) NLPA Density 4

Figure 8: Analysis on Local Group Changes

To further understand the performance of t-opt and k-opt, we provide analysis on local group changes. A local group change is a successful assignment implementation in an optimization group. For every group change, we collect the quality gain and the number of variables that were locked for implementing the new assignment. We plot all group changes for each set of graphs in a scatter chart, where a data point represents a **(Gain, #Locked Variables)** pair. The efficiency of an algorithm can be measured by the distribution of all group changes. Since larger lock sizes indicate higher chance of conflicts, group changes that have a large gain but small number of variables locked are most favored. The results on random graphs are shown in Figure 8(a) and Figure 8(b). We can see, $t = 1$ often has a larger gain but needs to lock more variables than $k = 3$. In particular, while $k = 3$ never identifies a gain larger than 70,000 and barely locks more than 20 variables, $t = 1$ often finds group changes where the gain is greater than 70,000 and the number of locked variables is greater than 20. On scale free graphs, as shown in Figure 8(c) and Figure 8(d), $t = 1$ often finds larger gains than $k = 3$. Particularly, the largest gain of $t = 1$ is almost quadruple of that of $k = 3$. We also note $k = 3$ often locks a large set of variables. Nearly half of the group changes require to lock more than 20 variables. This explains why $t = 1$ outperforms $k = 3$ in both convergence speed and final quality on scale free graphs. Figure 8(e) and Figure 8(f) show the results on NLPA graphs, where $k = 3$ is the most inefficient because it often locks a huge number of variables but barely identifies large gains.

Figure 7(e) shows the effect of increasing values of t on density 2 random graphs (domain size 5, reward drawn uniformly randomly from 0 to 625). We see that the different levels of t offer increasing final quality but also increasing convergence time. We also note increasing t doesn't give consistent gain in final quality. While $t = 1$ is significantly better than $t = 0$, the gain of $t = 3$ over $t = 2$ is much smaller. We also tested the ability of our algorithms to scale to very large graphs. In Figure 7(d), we show the number of time units required to reach a percentage of the final quality for both *t-distance-optimality* and *k-size-optimality*. We see that increasing the size of the random graph by tenfold to 1000 nodes

barely increases the time necessary for convergence at all, showing impressive scalability. To show the improvements offered by the partial synchronization (PS) and subset locking (SL) methods, we tested the t-distance algorithm for $t = 1$ with different PS window sizes and without SL. As can be seen in Figure 7(f), both techniques improve performance in isolation and in tandem.

Tables 1(a), 1(b), and 1(c) give additional statistics about algorithm performance in the first set of experiments on density 4 graphs. "Msgs" is the average number of messages per agent per time step. "MsgSize" is the average message size per message. "Evals" is the average number of constraint evaluations per agent per time step, which is a rough measure of the computation burden on each node. "Conflicts" is the total number of failed lock attempts. All of these are accumulated for 500 time units. We see that our algorithm for *t-distance-optimality* is generally much more efficient than KOPT in message size and the number of messages sent out, but does place a somewhat higher computational burden on the nodes. *t-distance-optimality* generally requires more computation than *k-size-optimality* because of its larger group size. We also note that *k-size-optimality* generally sends fewer but larger messages, which may offer some additional benefits in real world implementations.

Table 1: Additional Statistics

(a) Statistics for scale-free density 4 graphs.

	Msgs	MsgSize	Evals	Conflicts
T 1	1.27	28.44	187.62	405.64
K 3	0.72	43.99	114.64	411.60
KOPT 3	3.12	2970.83	72.84	0.00

(b) Statistics for random density 4 graphs.

	Msgs	MsgSize	Evals	Conflicts
T 1	0.85	26.68	81.81	445.76
K 3	0.47	41.51	45.51	410.50
KOPT 3	3.20	1209.94	57.29	0.00

(c) Statistics for NLPA density 4 graphs.

	Msgs	MsgSize	Evals	Conflicts
T 1	2.55	36.52	270.65	300.72
K 3	1.90	46.01	338.76	358.54
KOPT 3	3.12	7093.90	100.42	0.00

6. CONCLUSION

We make three key contributions. First, we introduce the novel concept of *t-distance-optimality*, and establish solution quality bounds for this concept that are often tighter than known bounds for k-size optimality. Second, we develop asynchronous local search algorithms for *t-distance-optimality* that outperform existing synchronous algorithms for *k-size-optimality*. Finally, in our experimental evaluation we investigate the tradeoff between k and t optimality, showing that *t-distance-optimality* offers considerable advantages for some types of DCOP (notably scale free graphs), while *k-size-optimality* has advantages in others.

7. REFERENCES

[1] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[2] B. Bollobas. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

[3] E. Bowring, J. P. Pearce, C. Portway, M. Jain, and M. Tambe. On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In *AAMAS-08*, 2008.

[4] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1427–1429, New York, NY, USA, 2006. ACM.

[5] J. S. Cox, E. H. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 821–827, New York, NY, USA, 2005. ACM.

[6] B. Faltings, D. Parkes, A. Petcu, and J. Shneidman. Optimizing streaming applications with self-interested users using M-DPOP. In *COMSOC-06*, 2006.

[7] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer, 2003.

[8] H. Katagishi and J. P. Pearce. KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *DCR-07*, 2007.

[9] R. Mailler and V. Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *AAMAS-04*, 2004.

[10] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.

[11] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI-07*, 2007.

[12] J. P. Pearce, M. Tambe, and R. T. Maheswaran. Solving multiagent networks using distributed constraint optimization. *AI Magazine*, 29(3):47–66, 2008.

[13] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI-05*, 2005.

[14] W. Schreiner. A java toolkit for teaching distributed algorithms. In *ITCSE-02*, pages 111–115, 2002.

[15] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1–2):55–87, 2005.

Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs

M. Vinyals
IIIA, Artificial Intelligence Research Institute
Spanish National Research Council
meritxell@iiia.csic.es

J.A. Rodriguez-Aguilar
IIIA, Artificial Intelligence Research Institute
Spanish National Research Council
jar@iiia.csic.es

J. Cerquides *
WAI, Dep. Matemàtica Aplicada i Anàlisi
Universitat de Barcelona
cerquide@maia.ub.es

ABSTRACT

In this paper we propose a novel message-passing algorithm, the so-called Action-GDL, as an extension to the Generalized Distributive Law algorithm (GDL) [1] to efficiently solve DCOPs. We show the generality of Action-GDL by proving that it has DPOP, one of the low-complexity, state-of-the-art algorithm to solve DCOPs, as a particular case. Finally, we provide empirical evidences to illustrate how Action-GDL can outperform DPOP in terms of computation, communication and parallelism needed to solve the problem.

1. INTRODUCTION

Multi-agent Coordination Problems (MCPs), also called distributed multi-agent decision making problems, are a class of problems in MAS focusing on how to coordinate agents' actions in order to yield a global desired behaviour for the MAS. Distributed Constraint Optimization Problems (DCOPs) are an extension of Constraint Optimization Problems (COPs) that can model a large class of MCPs [9].

State-of-the-art complete algorithms to solve DCOPs adopt two main approaches: search and dynamic programming. Search algorithms, like ADOPT [7], require linear-size messages, but an exponential number of messages. Dynamic programming algorithms, represented by the DPOP algorithm and its extensions [10], only require a linear number of messages, but their complexity lies on the message size, which may be very large.

In this paper, we formulate a new algorithm, the so-called Action-GDL, that takes inspiration from the GDL algorithm [1], extending and applying it to DCOPs. GDL is a general message-passing algorithm that exploits the way a global function factors into a combination of local functions generalizing a large family of well-known algorithms (e.g. Viterbi's, Pearl's belief propagation, or Shafer-Shenoy algorithms). Therefore, GDL has a wide range of applicability. In our case, the rationale to apply (and extend) GDL is that a DCOP requires the maximization of a global function resulting from the combination of local functions. In order to ensure optimality and convergence GDL must arrange the global function to optimise into a junction tree structure (JT) [5].

*Partially funded by TIN2006-15662-C02-01. M.Vinyals is supported by the Ministry of Education of Spain (FPU grant AP2006-04636). JAR thanks JC2008-00337.

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

There are several works that have applied JTs (join trees or tree-decompositions) to the constraint optimization problem [3, 4, 6]. Indeed the cluster-tree elimination algorithm [6] is an instance of GDL algorithm applied over a tree decomposition (junction tree). However, to the best of our knowledge, all these approaches construct a JT using triangulation methods [5], which are not suitable when applied to problems that are distributed by nature because they produce JTs disreading the structure of the problem.

Therefore, one of our contributions in this paper is the Action-GDL algorithm that extends GDL by: (1) supporting the distribution of the problem using distributed junction tree structure (DJT) that maps cliques to agents; and (2) modifying the original GDL algorithm to create two phases which allows to reduce the size of one half of the messages.

To generate DJTs, we introduce the Distributed Junction Tree Generator (DJTG) algorithm at the pre-processing phase of Action-GDL. DJTG is a message-passing algorithm, based on the one formulated in [8], that allows agents to *distributedly* compile a DJT keeping any distribution of relations among agents. Therefore, DJTG creates a DJT that adapts to the underlying distributed nature of the problem. We will show how although finding the best junction tree has been shown to be NP-hard [5], this algorithm is general enough to exploit existing distributed heuristics in the literature [9, 2] to produce a DJTG to fed into Action-GDL.

Thereafter, we show that Action-GDL generalises DPOP, one of the low complexity, state-of-the-art algorithm to solve DCOPs. To do so, we: (1) prove that DPOP is a particular case of Action-GDL; and (2) show how Action-GDL can exploit DJTs as a more general structure to generate executions that cannot be achieved by DPOP via pseudotrees. Therefore, Action-GDL can efficiently solve DCOPs. To the best of our knowledge, we are the first in comparing and providing a mapping between the space of DJTs and the space of pseudotree arrangements. Finally we provide empirical evidence to show that the generality of Action-GDL can be exploited to outperform DPOP in terms of amount of computation, communication, and parallelism.

This paper is structured as follows. Firstly, we provide a definition of DCOP (section 2) and the notation we will use through out this paper (section 3). Section 4 introduces Action-GDL, as well as its connection with GDL and the DJTG algorithm that allow agents to compile the DCOP problem into a DJT. Then, in section 5 we show how Action-GDL extends DPOP, one of the state-of-the-art algorithms to solve DCOPs. Next, section 6 provides some evidences of how to exploit the generality of Action-GDL by showing how it can outperform DPOP when solving the same DCOPs. Finally, section 7 draws some conclusions and outlines paths for future research.

2. OVERVIEW OF DCOPS

Distributed Constraint Optimization Problem (DCOP) are an extension to the Constraint Optimization Problem (COP) that can model a large class of MCPs [9]. These problems consist of a set of variables, each one taking on a value out of a finite discrete domain. Each constraint in this context has a set of variables as input specifying a cost, namely a relation. The goal of a COP algorithm is to assign values to these variables so that the total utility is maximized. A DCOP [10, 7] is an extension to a COP where variables are distributed among agents.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Let $r : \mathcal{D}_r \rightarrow \Re^+$, where $\mathcal{D}_r$ is the projection of the joint domain space $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ over variables in the domain of *r*, be a *utility relation* that assigns a utility value to each combination of values of its domain variables. Formally, a DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \alpha \rangle$ where: $\mathcal{A}$ is a set of agents; $\mathcal{X}$ is a set of variables; $\mathcal{D}_n$ is the joint domain space for all variables; $\mathcal{R} = \{r_1, \ldots, r_p\}$ is a set of utility relations; and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to some agent.

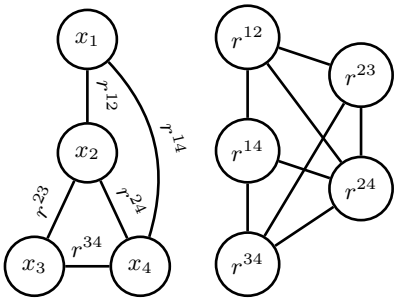
The objective function f is described as an aggregation (typically addition) over the set of relations. Formally:

$$f(d) = \sum_{i=1}^{p} r_i(d_{r_i}) \quad (1)$$

where d is an element of the joint domain space $\mathcal{D}$ and d_{r_i} is an element of $\mathcal{D}_{r_i}$. Solving a DCOP amounts to choosing values for the variables in $\mathcal{X}$ such that the objective function is maximized (minimized).

In a DCOP each agent receives knowledge about all relations that involve its variable(s) in addition to their domains. In general, DCOP algorithms do not impose any restriction regarding the number of variables that can be assigned to each agent or the arity of the relations. However, although all algorithms we refer to in this paper can deal with n-ary relations, for the sake of simplicity we mainly restrict them to unary and binary relations. Therefore, we will refer to unary relations involving variable $x_i \in \mathcal{X}$ as r^i, and to binary relations involving variables $x_i, x_j \in \mathcal{X}$ as r^{ij}.

A DCOP with binary relations is typically represented with its primal-constraint graph, whose vertices stand for variables and whose edges stand for binary relations, as shown by the example depicted in figure 1 (a). DCOPs can also be represented with its dual-constraint graph, whose vertices stand for relations and whose edges link relations that share some variable in their domains, as shown by the example depicted in figure 1 (b).



(a) Primal graph (b) Dual graph

Figure 1: Different representations for the same DCOP

3. NOTATION

Next we provide the definitions of a collection of functions and operators that we shall employ throughout the rest of this paper. Henceforth, given some variable set $X \subseteq \mathcal{X}$, $\mathcal{D}_X$ will stand for the joint domain space of variables in X. Furthermore, for exemplary purposes we assume that each domain $\mathcal{D}_i$ contains constant values $c_i^1, \ldots, c_i^{n_i}$.

DEFINITION 1 (DV). *The domain variable function DV returns the domain variables of a given set of relations.*
Ex: $DV(\{r^{31}\}) = \{x_3, x_1\}$, $DV(\{r^{31}, r^{34}\}) = \{x_1, x_3, x_4\}$.

DEFINITION 2 (COMPLEMENTARY VARIABLES). *Given a set of variables X and a relation r, we define the complementary variables of X by r as the set of variables in r that are not in X. Formally, $\bar{X}^r = DV(r) \setminus X$.*

DEFINITION 3 (UTILITY MESSAGE). *A message from agent a_i to agent a_j is a utility message over $X \subseteq \mathcal{X}$, if the information sent is a utility relation over $\mathcal{D}_X$. Henceforth, we shall denote that utility relation as μ_{ij}.*

DEFINITION 4 (ASSIGNMENT). *Given a set of agents $X \in \mathcal{X}$, an assignment σ over X sets a value to each variable $x_k \in X$ and sets free the remaining variables. Given $Y \subset X$, we note by $\sigma[Y]$ the projection of σ to Y, that is, the assignment that sets the same value as σ for the variables in Y.*
Ex: $X = \{x_1, x_2, x_3\}$, *σ an assignment over X,* $\sigma(x_1) = c_1^2$, $\sigma(x_3) = c_3^5$, *x_2 is free in σ*
$Y = \{x_1\}$, $\sigma[Y](x_1) = c_1^2$, *x_2 and x_3 are free in $\sigma[Y]$*

DEFINITION 5 (VALUE MESSAGE). *A message from agent a_i to agent a_j is a value message over $X \subseteq \mathcal{X}$ if the information sent is an assignment over X. Henceforth, we shall denote such assignment by σ_{ij}.*

The joint operator is a combination operator that joins the knowledge represented by two relations into a single one by adding their values.

DEFINITION 6 (JOINT). *Let r, s be two relations and $\mathcal{D}_{r \otimes s} = \times_{x_k \in DV(\{r,s\})} \mathcal{D}_k$ be their joint domain space. The combination of r and s (noted $r \otimes s$) is a utility relation over $\mathcal{D}_{r \otimes s}$ such that $(r \otimes s)(d) = r(d_r) + s(d_s)$ for all $d \in \mathcal{D}_{r \otimes s}$, where $d_r \in \mathcal{D}_r$ and $d_s \in \mathcal{D}_s$ are the projections of d over the domains of relations r and s respectively.*
Ex: $(r^{13} \otimes r^{14})(c_1^2, c_3^5, c_4^1) = r^{13}(c_1^2, c_3^5) + r^{14}(c_1^2, c_4^1)$.

We can readily generalize the joint operator over a finite set of relations: $\bigotimes_{\{r_1, \ldots, r_m\}} = r_1 \otimes (r_2 \otimes \ldots (r_{m-1} \otimes r_m) \ldots)$.
The projection operator sums up the utility that a relation contains over a set of variables. Thus, the projection operator over a relation r and a set of variables Xassesses the r maximum utility for the variables in X.

DEFINITION 7 (PROJECTION). *The projection operator of relation r over a set of variables X is a summarization operator that returns a utility relation over $\mathcal{D}_X$ such that* $(\bigoplus_X r)(d_X) = \max_{d_{\bar{X}^r} \in \mathcal{D}_{\bar{X}^r}} r(d_X, d_{\bar{X}^r})$.
Ex: $(\bigoplus_{\{x_3\}} r^{13})(c_3^2) = \max_{k \in \mathcal{D}_1} r^{13}(k, c_3^2)$.
Notice that we can employ the projection operator by specifying the variables to eliminate from a relation as follows $\bigoplus_{\setminus X} r = \bigoplus_{\bar{X}^r} r = \bigoplus_{DV(r) \setminus X} r$.

DEFINITION 8 (SLICE). *The slice of a relation r by an assignment σ over X is a utility relation over $\mathcal{D}_{\bar{X}^r}$ such that $(\underset{\sigma}{\triangledown} r)(d_{\bar{X}^r}) = r(d_X, d_{\bar{X}^r})$ where $d_X \in \mathcal{D}_X$ contains the values set by σ to the variables in X.*
Ex: $X = \{x_3\}$, $\sigma(x_3) = c_3^2$, $(\underset{\sigma}{\triangledown} r^{13})(c_1^1) = r^{13}(c_1^1, c_3^2)$.

4. THE ACTION-GDL ALGORITHM

In this section we introduce the Action-GDL, a novel complete algorithm to efficiently solve DCOP's, an extension to GDL [1] to efficiently apply it to MAS decision making. We start by introducing GDL in the following section to subsequently propose Action-GDL in section 4.2. Since Action-GDL is executed over a distributed junction tree structure, for completeness, we proposed it to be combined with what we called the DJTG algorithm, an algorithm that allow agents to distributedly compile JTs initially proposed in [8] in the context of sensor networks. DJTG algorithm (section 4.3) allows agents to distributedly compile the DCOP into a DJT to which Action-GDL is executed over and drawbacks that other traditional methods to compile DJTs have been reported to have in distributed environments.

4.1 The GDL Algorithm

GDL [1] is a general message-passing algorithm that exploits the way a global function factors into a combination of local functions to compute the objective function in an efficient manner. GDL is defined over two binary operations [1] that in our case, since we are concerned with the problem of maximizing an utility function, correspond to the addition and the maximization (the max-sum GDL). In order to ensure optimality and convergence, GDL arranges the objective function to assess in a *junction tree structure* (JT)[5].

DEFINITION 9. *A **junction tree** (JT) is a tree of cliques that can be represented as a tuple $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables; $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ is a set of cliques such that each clique $\mathcal{C}_i \subseteq \mathcal{X}$; $\mathcal{S}$ is a set of separators, where each separator is an edge between two cliques containing the intersection of the cliques[1]; and $\Psi = \{\psi_1, \ldots, \psi_m\}$ is a set of potentials, where potential ψ_i is a function assigned to clique $\mathcal{C}_i$ with domain $\Delta_i \subseteq \mathcal{X}$. Furthermore, the following properties must hold:*
• ***Single-connectedness.*** *Separators create exactly one path between each pair of cliques.*
• ***Covering.*** *Each potential domain is a subset of the clique to which it is assigned, namely $\Delta_i \subseteq \mathcal{C}_i$.*
• ***Running intersection.*** *If a variable x_i is in two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$, then it must also be in all cliques on the (unique) path between $\mathcal{C}_i$ and $\mathcal{C}_j$.*

Likewise variables in DCOP, we assume that the variables in a junction tree are defined over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Moreover, $\mathcal{D}_{\mathcal{C}_i}$ stands for clique $\mathcal{C}_i$ domain space, namely the joint domain space of the variables in clique $\mathcal{C}_i$.

Figure 2 shows a JT where circles stand for cliques, labelled with the variables each one contains, and edges between cliques stand for separators. Thus, for example, C_1 contains variables x_2, x_4; C_3 contains variables x_2, x_3, x_4; and their separator is composed of their intersection x_2, x_4. Each clique C_i is associated with a potential ψ_i, a function whose domain is a subset of C_i.

GDL defines a message-passing phase for cliques to exchange information about their variables. Once the message-passing phase is over, each clique can compute its state, namely its variables states. To illustrate the way the max-sum GDL operates, consider the following example. Say that our goal is to distributedly maximize some objective function $f(x_1, x_2, x_3, x_4) = \psi_1(x_2) + \psi_2(x_1, x_2, x_4) + \psi_3(x_2, x_3, x_4)$, whose factors ($\psi_1$, ψ_2 and ψ_3) are arranged in the directed JT of figure 2. Since the JT is directed, messages are sent in two different message-passing phases: (i) one up the tree in which each clique sends a message to its clique parent when, for the first time, it has received messages from all of its children; (ii) one down the tree so that each clique sends a message to its children when it receives a message from its parent.

[1]Formally, a separator s^{ij} between clique $\mathcal{C}_i$ and $\mathcal{C}_j$ is defined as $s^{ij} = \mathcal{C}_i \cap \mathcal{C}_j$.

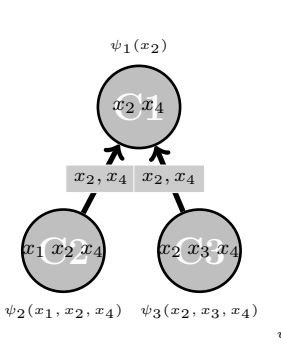


Figure 2: JT

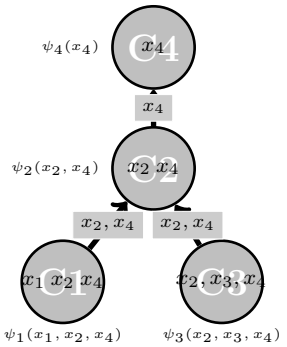


Figure 3: DJT

#	Message/local knowledge ($\widehat{\mathcal{K}}$)
1.	$\mu_{21}(x_2, x_4) = max_{\{x_1\}} \psi_2(x_1, x_2, x_4)$
2.	$\mu_{31}(x_2, x_4) = max_{\{x_3\}} \psi_3(x_2, x_3, x_4)$
3.	$\widehat{\mathcal{K}}_1(x_2, x_4) = \psi_1(x_2) + \mu_{21}(x_2, x_4) + \mu_{31}(x_2, x_4)$
4.	$\mu_{12}(x_2, x_4) = \psi_1(x_2) + \mu_{31}(x_2, x_4)$
5.	$\mu_{13}(x_2, x_4) = \psi_1(x_2) + \mu_{21}(x_2, x_4)$
6.	$\widehat{\mathcal{K}}_2(x_1, x_2, x_4) = \psi_2(x_1, x_2, x_4) + \mu_{12}(x_2, x_4)$
7.	$\widehat{\mathcal{K}}_3(x_2, x_3, x_4) = \psi_3(x_2, x_3, x_4) + \mu_{13}(x_2, x_4)$

Table 1: Trace of GDL over the JT of Fig. 2

At round 1, clique $\mathcal{C}_2 = \{x_1, x_2, x_4\}$ sends a message μ_{21} to clique $\mathcal{C}_1 = \{x_2, x_4\}$ with the values of its local function, ψ_2, after 'filtering out' dependence on all variables but those common to $\mathcal{C}_2$ and $\mathcal{C}_1$ (namely variables which are not in their separator). At round 3, after clique $\mathcal{C}_1$ receives the values of its children's local functions for its variables x_2, x_4, it combines those values into $\widehat{\mathcal{K}}_1$. $\widehat{\mathcal{K}}_1$ is a function that stands for $\mathcal{C}_1$ knowledge over its variables, namely x_2, x_4. At that point, since $\mathcal{C}_1$ has received messages from all its neighbors, $\widehat{\mathcal{K}}_1$ contains all the information related to x_2, x_4. At rounds 4 and 5, clique $\mathcal{C}_1$ sends messages to its children that contain the combination (joint operation) of its local function, ψ_1, with other children messages. Thus, $\mathcal{C}_2$ receives a message from $\mathcal{C}_1$ that contains the potential ψ_1 combined with μ_{31}. Then it can compute $\widehat{\mathcal{K}}_2$(round 6).

4.2 Extending GDL to solve DCOPs

Recall that our goal is to solve MCPs represented as DCOPs. Therefore, the capability of computing any objective function, as provided by GDL, is not enough. We need to go one step beyond GDL to allow a group of agents make a joint decision (regarding their variables' values) that maximizes any objective function. For this purpose, Action-GDL extends GDL by: (1) inferring decision variables; and (2) supporting the distribution of the problem through the use of a distributed junction tree structure.

Inferring decision variables Consider a DCOP setting. As explained above in GDL, when a clique has received messages from all its neighbors, it has all information related to its variables and it can compute its objective function. In DCOPs, clique variables are decision variables and computing a clique objective function stands

Algorithm 1 Action-GDL($\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$)

Each agent $a \in \mathcal{A}$ for each one of its cliques $\mathcal{C}_i$ starts with $\langle \widehat{P}(\mathcal{C}_i), \widehat{Ch}(\mathcal{C}_i), \mathcal{C}_i, \psi_i, \widehat{S}(\mathcal{C}_i), \beta_i \rangle$ and runs:

1: **Phase I: UTILITY Propagation**
2: $\widehat{\mathcal{K}}_i = \psi_i$
3: **for all** $\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)$ **do**
4: Wait for utility message μ_{ji} from $\mathcal{C}_j$'s agent (that is $\beta_i(\mathcal{C}_j)$)
5: $\widehat{\mathcal{K}}_i = \widehat{\mathcal{K}}_i \otimes \mu_{ji}$
6: **end for**
7: **if** $\mathcal{C}_i$ is not the tree's root, let $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ **then**
8: Let $s^{ip} \in \widehat{S}(\mathcal{C}_i)$ be the separator between i and its parent
9: Send $\mu_{ip} = \bigoplus_{s^{ip}} \widehat{\mathcal{K}}_i$ to $\mathcal{C}_p$'s agent (that is $\beta_i(\mathcal{C}_p)$)
10: **end if**
11: **Phase II: VALUE propagation**
12: **if** $\mathcal{C}_i$ is not the tree's root, let $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ **then**
13: Wait for a value message σ_{pi} from $\mathcal{C}_p$'s agent (that is $\beta_i(\mathcal{C}_p)$)
14: $\widehat{\mathcal{K}}_i = \bigtriangledown_{\sigma_{pi}} \widehat{\mathcal{K}}_i$; /*Slice $\widehat{\mathcal{K}}_i$ with the value message*/
15: **end if**
16: $d^* = arg \max_{d \in \mathcal{D}_{DV(\widehat{\mathcal{K}}_i)}} \widehat{\mathcal{K}}_i$; /*Fix the values for the free variables*/
17: $d^*_{\mathcal{C}_i} = d^* \cup \sigma_{pi}$; /*Put together the assesed values and the value message received. Assume the root gets an empty value message*/
18: **for all** $\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)$ **do**
19: Let $\sigma_{ij} = d^*_{\mathcal{C}^i}[s_{ij}]$; /*Project into the separator*/
20: Send σ_{ij} to $\mathcal{C}_j$'s agent (that is $\beta_i(\mathcal{C}_j)$)
21: **end for**
22: **return** $d^*_{\mathcal{C}_i}$;

for assigning values to these decisions. Therefore, when a clique infers their state, there is no need to propagate more information related to its variables since we can propagate directly the decisions taken. In other words, there is no need to propagate messages containing relations down the tree because all a child requires to make a decision is its father's decisions (variables' assignments). It implies that in a DCOP, when the first message-passing phase of GDL, up to the tree, is over, the second message-passing phase of GDL, down the tree, is no longer necessary. Thus, we require a second message-passing phase for cliques to exchange *decisions* down the tree, which is precisely the extension that Action-GDL introduces. Henceforth, we shall refer to the first message-passing phase as *utility propagation*, and to the second one as *value propagation*. It is relevant to notice that the value propagation phase ensures that whenever multiple optimal joint decisions are feasible, cliques converge to the very same joint decision, namely to the very same solution of a DCOP.

Table 2 displays a trace of Action-GDL over the JT in figure 2. Notice that by making $\Psi = \{\psi_1 = r^2, \psi_2 = r^{12} \otimes r^{14} \otimes r^{24}, \psi_3 = r^{23} \otimes r^{34}\}$ the function encoded in the JT ($f(x_1, x_2, x_3, x_4) = \psi_1(x_2) + \psi_2(x_1, x_2, x_4) + \psi_3(x_2, x_3, x_4)$) of is the same as the constraint graph of figure 1(a), so we are maximizing a DCOP function.

Steps 1-4 are equivalent to steps 1-3 in GDL. However, at step 5 the root clique assesses the optimal value for x_2, x_4 ($x_2^* = c_2^*, x_4^* = c_4^*$) and propagates these values down the tree through value messages to cliques $\mathcal{C}_2$ and $\mathcal{C}_3$ (steps 6 and 7). At steps 8-9 and 10-11, $\mathcal{C}_2$ and $\mathcal{C}_3$ assess the values of x_1 and x_3, respectively, using its parent decision values (c_2^*, c_4^*).

Supporting the distribution of the problem. Another major difference between Action-GDL and GDL has to do with the way they solve a problem. GDL runs over a JT as formalised by definition 9. Hence, all cliques are considered to be located in a single agent, which is in charge of running GDL. Action-GDL solves a

#.	Messages/local knowledge $\widehat{\mathcal{K}}$	#.	Messages/local knowledge $\widehat{\mathcal{K}}$
1.	$\mu_{21}(x_2, x_4)$=$\max_{\{x_1\}} \psi_2(x_1, x_2, x_4)$	7.	$\sigma_{13}(x_2, x_4)$=(c_2^*, c_4^*)
2.	$\widehat{\mathcal{K}}_1(x_2, x_4)$=$\psi_1(x_2) + \mu_{21}$	8.	$\widehat{\mathcal{K}}_2(x_1)$=$\psi_2(x_1; \sigma_{12})$
3.	$\mu_{31}(x_2, x_4)$=$\max_{\{x_3\}} \psi_3(x_2, x_3, x_4)$	9.	c_1^*=$arg \max_{\{x_1\}} \widehat{\mathcal{K}}_2$
4.	$\widehat{\mathcal{K}}_1(x_2, x_4)$=$\widehat{\mathcal{K}}_1(x_2, x_4) + \mu_{31}$	10.	$\widehat{\mathcal{K}}_3(x_3)$=$\psi_3(\sigma_{13}; x_3)$
5.	(c_2^*, c_4^*)=$arg \max_{\{x_2, x_4\}} \widehat{\mathcal{K}}_1$	11.	c_3^*=$arg \max_{\{x_3\}} \widehat{\mathcal{K}}_3$
6.	$\sigma_{12}(x_2, x_4)$=(c_2^*, c_4^*)		

Table 2: Trace of Action-GDL over the JT of Fig. 2

DCOP where variables and relations are distributed over agents that cooperatively solve the problem. Therefore, Action-GDL extends GDL to deal with cliques that are distributed to different agents and control that agents have knowledge about the local information (potential) related to its cliques. This is accomplished by running Action-GDL over a *distributed junction tree* (DJT). Formally:

DEFINITION 10. *A **distributed junction tree** (DJT) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$ where $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ is a JT; $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a set of agents; and $\beta : \mathcal{C} \rightarrow \mathcal{A}$ maps each clique to one agent.*

We define $\widehat{N}(\mathcal{C}_i) = \{\mathcal{C}_j | s^{ij} \in \mathcal{S}\}$ as a function that returns the cliques connected by a separator to clique $\mathcal{C}_i$, namely its neighboring cliques. Since a DJT is a tree of cliques it can also be defined as a directed tree. In a directed DJT we define two additional relationships among cliques: $\widehat{P}(\mathcal{C}_i)$, which returns the parent of $\mathcal{C}_i$; and $\widehat{Ch}(\mathcal{C}_i)$, which returns the children of $\mathcal{C}_i$.

An example of DJT is given in figure 3. Likewise JTs, circles stand for cliques, and edges for separators, both labelled with their variables' indices. This DJT has 4 cliques, one for each agent of the DCOP of figure 1(a) (clique C_i is assigned to agent a_i). The set of potentials contains the set of relations of the DCOP distributed as follows: $\psi_1 = r^{12} \otimes r^{14}$,$\psi_3 = r^{23} \otimes r^{34}$,$\psi_2 = r^{24}$,$\psi_4 = \{\}$,$\psi_5 = r^{15}$. Notice that this DJT has the property that agents are assigned a clique whose potential contains relations that this agent knows (in DCOP relations that contains some agent's variable). Thus, agent 1 is assigned clique 1 whose potential contains relations that include variable x_1, namely r^{12}, r^{14}. That is not true in the JT of figure 2 since in that case there is not a single agent who knows all relations assigned to potential ψ_2, namely r^{12}, r^{14}, r^{24}.

Algorithm 1 outlines Action-GDL. Given a DJT $\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$, each agent $a \in \mathcal{A}$ involved in Action-GDL, only needs to know the subset of the DJT that involves the cliques it has assigned. Hence, every agent is assumed to start knowing a tuple $\langle \widehat{P}(\mathcal{C}_i), \widehat{Ch}(\mathcal{C}_i), \psi_i, \widehat{S}(\mathcal{C}_i), \beta_i \rangle$ for each one of its cliques $\mathcal{C}_i$, where $\widehat{\mathcal{S}}$ returns a clique's separators ($\widehat{\mathcal{S}}(\mathcal{C}_i) = \{s^{ik} | s^{ik} \in \mathcal{S}\}$), and β_i returns the agents assigned to clique $\mathcal{C}_i$'s parent or children.

During the utility propagation phase (lines 1-10), agents exchange utility messages. The initial knowledge for each clique is its potential (line 2). For each clique, its agent waits until receiving a utility message from each of its children cliques (lines 3-4). These messages contain a utility relation over the variables shared by both cliques (their separator) and are sent by agents assigned to the children cliques. Every time that the agent receives a new utility message, it incorporates it (by using the combination operator) to its local knowledge (line 5). After combining utility messages from all the children of a clique, if that clique has a parent (line 7), its agent summarizes that part of its local knowledge (using the projection operator) that is of interest to the clique's parent (by means of a utility relation over its separator) and sends it to the agent associated to the parent's clique (line 9).

During the value propagation phase (lines 11-21), agents compute the optimal values for their variables and exchange value messages, namely decisions. Given a clique, its agent waits until receiving a value message (containing value assignments) for all variables in common (in the separator) with its clique parent (line 12-

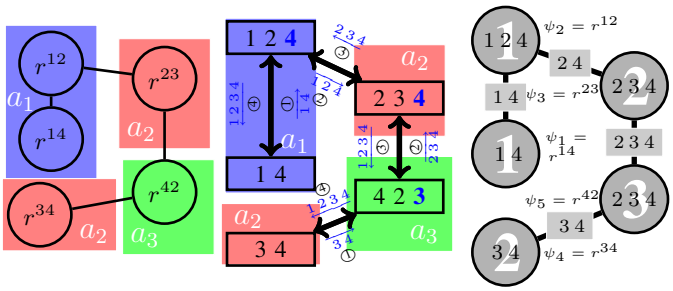
13). At that point, the agent has received all the knowledge, in form of utility (from children) and value (from the parent) messages, required for computing the objective function related to its clique variables. The agent slices its knowledge by incorporating the already inferred decisions (line 14) and computes the optimal values for the rest of its clique variables (line 16). Once an agent knows the variables' values for one of its cliques, it can propagate them down the tree (lines 18-21). Notice however that it only propagates variable assignments that are required by its children cliques, namely assignments for variables in their separator.

Since Action-GDL runs over a DJT, we can readily assess its computation and communication complexity from cliques' and separators' sizes after [1, 8]. Action-GDL requires a number of messages linear to the number of edges in the DJT (exchanging one value message and one utility message per separator). The communication complexity lies in the size of utility messages, which is exponential to separators' sizes, because the size of value messages is linear. Regarding the computation required by each agent to build messages and assess variables' values, it also scales with its cliques' sizes.

4.3 The DJTG algorithm

As explained above, Action-GDL runs over a DJT (as given by definition 10). It has been argued [9] that traditional methods to compile JTs, that is triangulation methods based on the one proposed in [5], are not suitable when applied to problems that are distributed by nature because they produce JTs disregarding the structure of the problem. Thus, cliques in such methods are created independently of the number of agents and their knowledge and therefore since the mapping agent-clique is not clear it can appear a clique that has knowledge about a particular relation/variable that it's hosting agent does not know. Therefore, here we propose to use an alternative algorithm, the so-called Distributed Junction Tree Generator (DJTG) that distributedly compiles a DJT, by exchanging a linear number of messages, that captures the distribution of relations required by the problem. Such DJT can be readily fed into, and hence solved by Action-GDL.

The DJTG algorithm receives as input a set of relations distributed among agents and an spanning tree, $\mathcal{ST}$, defined over them. Figure 4(a) illustrates an input to DJTG. Observe that relation r^{12}



(a) Spanning tree (b) DJTG execution (c) Junction tree

Figure 4: DJTG execution

is assigned to agent a_1 and it is linked to relations r^{23} and r^{14} by edges in the $\mathcal{ST}$.

DJTG has two phases: 1) a pre-processing phase where agents create a distributed clique tree that may not satisfy the running intersection property (RIP); followed by 2) a message-passing phase that calculates the unique set of minimal cliques that satisfy the RIP.

In the pre-processing phase, each agent a creates a clique $\mathcal{C}_j$ for each one of its relations r_j and sets as potential the relation itself, namely $\psi_j = r_j$. Cliques are initially set to their potential domain, namely $C_i = \Delta_i$, in order to readily ensure the covering property. Moreover, for every two relations r_i, r_j connected in the $\mathcal{ST}$, agents create a separator s^{ij} linking their corresponding cliques $\mathcal{C}_i$ and $\mathcal{C}_j$. Figure 4 (b) shows the structure produced by the pre-processing phase. Boxes stand for cliques containing numbers that stand for variables' indices. Cliques are assigned to agents. For instance, clique $\mathcal{C}_1$ with variables x_1, x_4 and clique $\mathcal{C}_2$ with x_1, x_2 are assigned to a_1. The variables correspond to the domains of the cliques' potentials, namely the domains of r^{14} and r^{12} respectively. Cliques $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected as their relations r^{14} and r^{12} in the $\mathcal{ST}$.

The second phase of DJTG is responsible for ensuring the RIP. In that phase, each agent exchanges for each one of its cliques, $\mathcal{C}_i$, *reachability* messages with agents related to $\mathcal{C}_i$'s neighbors that contain the set of reachable variables from $\mathcal{C}_i$. Figure 4(b) shows the messages exchanged. Single-directed arrows between boxes stand for messages exchanged between cliques. Each arrow is labelled with some variables' indices and a circled number standing for the order of the message in the message-passing execution. The set of reachable variables from a clique $\mathcal{C}_i$ to $\mathcal{C}_j$ is calculated as the union of: (i) $\mathcal{C}_i$'s potential domain; and (ii) the variables reachable from $\mathcal{C}_i$'s neighbours other than $\mathcal{C}_j$. Thus, agent a_1 sends a message to agent a_2 for clique $\mathcal{C}_3$ that contains variables (x_1, x_2, x_4), namely the variables that can be reached from clique $\mathcal{C}_2$. These variables are the result of the union of $\mathcal{C}_2$ potential domain, namely (x_1, x_2), with the reachable variables from $\mathcal{C}_2$, namely (x_1, x_4). Once an agent receives, for a given clique, reachability messages from all its neighbours, it redefines its clique adding variables that are in more than one *reachability* message. In figure 4(b) agent a_2 receives two reachability messages for clique $\mathcal{C}_3$: one with (x_1, x_2, x_4) from clique $\mathcal{C}_2$ associated to a_1, another one with (x_2, x_3, x_4) from clique $\mathcal{C}_5$ associated to a_3. Since both messages contain x_4, agent a_2 knows that its clique $\mathcal{C}_3$ must also carry x_4 to satisfy the RIP. After computing cliques, it is straightforward to assess separators (see definition 9). Finally, figure 4(c) depicts the DJT as produced by DJTG from the initial distribution of relations in figure 4(a). Notice that by creating a clique per relation and by assigning each clique to the agent associated to that relation, DJTG manages to preserve the initial distribution of the problem.

This alternative way of building a JT, by directly ensuring the RIP over a set of relations was initially formulated in [8] in the context of sensor networks. However, they restricted each agent to control a single clique whose potential results from the combination of relations located to the agent. DJTG extends the algorithm in [8] to: (1) allow each agent to be associated to more than one clique; and (2) accept as input a spanning tree defined over some set of relations, without making any assumptions on their composition.

Given a set of n relations, there are n^{n-2} different spanning trees that we can define over them, and for each one we can compile the associated DJT with the DJTG algorithm. It is know from [5] that finding the optimal JT is NP-hard, so it is reasonably to wonder what we can do to find good spanning trees to use as input for the DJTG algorithm. However it turns out that existing heuristics proposed in the literature for DCOP problems and DJT construction can be expressed, explicitly or implicitly, as a set of relations connected by a spanning tree that we can use as input of the DJTG. On the one hand, there are heuristics [8] that directly assess an spanning tree defined over the dual-constraint graph and we can readily exploit them. On the other hand, there are heuristics that define a spanning tree, or a subclass of them like pseudotrees, over the primal-constraint graph such as those proposed in [9, 2]. These heuristics associate each relation to the lowest variable of its do-

main in the tree structure. We can combine the relations associated to the very same variable to create a single relation. Since in these approaches variables are connected by an spanning tree, so are the combined relations and we can use this spanning tree as input to the DJTG.

5. GENERALITY OF ACTION-GDL

Action-GDL has a lot of straightforward similarities with DPOP [10], one of the state-of-the-art algorithms to solve DCOPs. Indeed DPOP was inspired by the sum-product algorithm, a GDL iterative version when applied directly to the original constraint graph and that is only guarantee convergence and optimality in acyclic graphs. DPOP ensures optimality and convergence in general graphs by arranging the DCOP into a pseudotree whereas Action-GDL arranges the problem in a DJTs.

Both algorithms, Action-GDL and DPOP have two similar phases when agents exchange same type of messages (UTIL and VALUE phases) and are based on the same operators: the combination and the projection operators. Despite of all these similarities, there are a set of important differences among them [2] namely:

(1) DPOP requires the DCOP to be arranged into a pseudotree structure with a particular distribution of relations (relations are associated to the lowest variable of its domain in the pseudotree) whereas Action-GDL is executed over a DJT. Figure 5(a) illustrates a pseudotree for the DCOP of figure 1(a). By definition of pseudotree[10], variables in different branches can not have direct dependencies among them. Thus, variables x_1 and x_3 are independent (there is no relation r^{13}) and relations r^{12},r^{14} are placed on x_1 (agent 1).

(2) In DPOP when agents create an utility message they use the summarize operator by summarizing out its own variable whereas in Action-GDL agents summarize over variables in the separator. Thus, during DPOP execution agent 1 (assigned to x_1) *filters out* x_1 from its local factor r^{12},r^{14} sending a message to agent 2 (assigned to x_2) that depends on the remaining variables, namely x_2 and x_4.

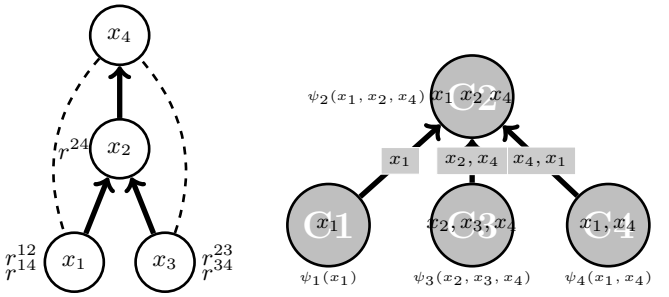
(3) In DPOP variables are inferred always one by one, in their position in the pseudotree, whereas in Action-GDL multiple variables can be inferred at once in the first agent that contains all the information related to these variables. Thus, during DPOP execution over the pseudotree of figure 5(a), agent 4 (responsible of x_4) infers its variable sending its value down to the tree. Same applies for the rest of variables.

However, if we want to compare DPOP and Action-GDL when applied to the same DCOP, we have to use equivalent arrangements for both algorithms. Thus we need to define a mapping between pseudotrees and DJTs. We will prove that given a pseudotree we can always define its equivalent DJT such that the execution of Action-GDL over such DJT is equal to the execution of DPOP over the pseudotree arrangement.

For example, the equivalent DJT of the pseudotree depicted in figure 5(a) is the DJT of figure 3. Notice that in that DJT relations are placed exactly as in the pseudotree arrangement and when cliques send messages summarizing over the separator, they *filter out* a single variable. Thus, DPOP execution over this pseudotree arrangement and Action-GDL execution over the DJT are equal.

However, it turns out that the other way around, that given a DJT we can always define an equivalent pseudotree arrangement,

[2]Here we list a set of similarities and differences respect to DPOP. We refer the reader to [10][9] for a detailed description of the DPOP algorithm



(a) Pseudotree (b) DJT with non equivalent pseudotree

Figure 5: Example of a pseudotree and DJT with non-corresponding pseudotree arrangement for the DCOP depicted in figure 1

is not true. Thus, given an Action-GDL execution over a DJT, the equivalent pseudotree may not exist. It is because, given a DCOP, the space of all possible pseudotrees arrangements map with a subspace of all possible DJTs. DPOP equations are a particular case of Action-GDL equations when it is executed over this subclass of DJTs. We illustrate this with an example. Take the DJT of figure . By making $\Psi = \{\psi_1 = , \psi_2 = r^{12} \otimes r^{24}, \psi_3 = r^{23} \otimes r^{34}, \psi_4 = r^{41}\}$ the function encoded is the same as the constraint graph of figure 1(a). However, this DJT can not have any equivalent pseudotree because multiple variables are eliminated at once (clique 2 infers variables x_1,x_2 and x_3) and there are cliques that do not eliminate any variable when summarizing over the separator, namely C_1 and C_4.

In what follows we provide an sketch of the proof of the equivalence between Action-GDL and DPOP (fully detailed in [11]). Next, in section 6 we will provide empirical evidence of how we can exploit DJTs, as a more general structure, to improve the problem solving cost with respect to DPOP.

5.1 Proving equivalence

In order to prove equivalence between Action-DGL and DPOP, we first define a mapping, which we shall name γ, that builds a DJT from each pseudotree. Then we prove (lemma 1) that both the computation performed and the messages exchanged during the utility propagation phase are the same. After that, we prove (lemma 2) that the messages exchanged during the value propagation phase are also the same. Finally, we prove that the algorithm DJTG can compute this mapping distributedly. Because of lack of space, in the following we just expose the results and sketching proofs. The interested reader can find far more detailed proofs in [11].

LEMMA 1. *Given a DCOP* Φ *and a pseudotree* PT*, the computation performed and the messages exchanged by each agent during the utility phase of* $DPOP(\Phi,\text{PT})$ *and Action-GDL(*$\gamma(\Phi, \text{PT})$ *) are the same.*

PROOF. We prove the lemma by induction on the depth of the agent in the pseudotree. Both in the base and induction cases, we can prove that: (i) the set of variables handled by agents in both algorithms are the same; and (ii) the domain of the utility messages send by agents in DPOP after eliminating its corresponding variable coincides with separators in Action-GDL. By induction the utility messages received by each agent in both algorithms are the same. This fact along with (i) and (ii) forces that the computation performed and messages exchanged during this phase by each agent must be the same. □

LEMMA 2. *Given a DCOP* Φ *and a pseudotree* PT *the value assigned by each agent to its variable and the messages exchanged during the value propagation phase of* $DPOP(\Phi,\text{PT})$ *and Action-GDL(*$\gamma(\Phi, \text{PT})$ *) are the same*

PROOF. We prove the lemma by induction on the depth of the pseudotree. The base case is trivial since there is only one variable in the pseudotree and both algorithms compute the same value for it. In the induction case we can split our pseudotree into the root and a set of pseudotrees of smaller depth. Then: (i) it is easy to see that the root agent acts equivalently in DPOP and in Action-GDL; and (ii) we can apply the induction hypothesis to the pseudotrees of smaller depth. Our result comes from (i) and (ii). □
Lemmas 1 and 2 prove the main result of this section:

THEOREM 1. *Given a DCOP* Φ *and a pseudotree* PT*, the execution of DPOP(*Φ*,* PT *) is equivalent to Action-GDL(*$\gamma(\Phi,$ PT *)).*

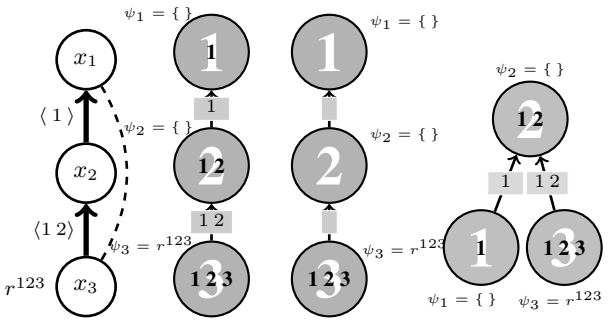
Theorem 1 shows that Action-GDL can be at least as efficient as DPOP in any DCOP problem. In the next section we introduce the DJTG algorithm that distributedly computes mapping γ at a cost that is small with respect to the cost of solving the problem.

THEOREM 2. *Given a DCOP* Φ *and a pseudotree* $\langle P, PP\rangle$*, the DJTG algorithm creates the DJT given by* $\gamma(\Phi, \langle P, PP\rangle)$

Therefore, DJTG computes the mapping γ at a cost that is small with respect to the cost of solving the problem. This two results prove that Action-GDL can be at least as efficient as DPOP (by mimicking its behavior).

6. EXPLOITING ACTION-GDL

At this point we have learned that Action-GDL generalises DPOP. It is now reasonable to wonder about the benefits that such generality delivers. In what follows we argue that Action-GDL can yield better algorithmic performance than DPOP. Action-GDL can achieve such improvement because: (i) DJTs allow to explore problem arrangements that cannot be represented via pseudotrees; and (ii) it can assess multiple variables' values at once. To show the benefits of Action-GDL with respect to DPOP, we empirically compare the computation and communication costs of both algorithms when solving the same DCOP. Moreover, we also compare the maximum degree of parallelism each algorithm can achieve.



(a) DPOP (b) DJT γ (c) Modified DJT (d) DJT rooted at 2

Figure 6: Example of experimented rearrangements

Our first experiment is aimed at showing the communication and computation savings achieved by Action-GDL with respect to DPOP. Such savings are obtained by adequately transforming the problem arrangement represented by a pseudotree. Consider the example depicted in figure 6. Figure 6(a) shows a pseudotree for a DCOP composed of a single ternary relation r^{123}. Observe that although variables x_1, x_2 do not have any relation, since DPOP can only eliminate variables one by one, its execution would propagate utility messages over these variables. Figure 6(b) depicts the DJT produced by mapping γ when applied to the pseudotree. Hence, according to theorem 1 the execution of Action-GDL over this DJT and the execution of DPOP over the pseudotree are equivalent. However, we can further transform the DJT in figure 6(b) to obtain savings. Notice that to make Action-GDL and DPOP executions equivalent,the definition of mapping γ (from pseudotree to DJT) *forces* that each clique in the equivalent DJT contains its variable although it is not part of its potential domain. If we do not enforce such constraint, the DJTG algorithm generates the DJT of figure 6(c), which can be regarded as a rearrangement of the one in figure 6(b). When running over the DJT in figure 6(c), Action-GDL does not need to exchange any utility messages, reducing the computation required to solve the problem. Hence there is a rationale for the rearrangement that we propose. Notice that when running Action-GDL, a variable' value is assessed at the clique that concentrates all its information. In figure 6(c), the values of x_2, x_3 are assessed at the clique containing x_1, x_2, x_3. That clique is in charge of propagating its decisions. Hence, there is no need to propagate utility messages involving x_1 and x_2 up the tree.

Next we compare the size of the messages exchanged and the amount of computation required by Action-GDL and DPOP when solving the same DCOP as the number of variables grows. Given a number of variables $n \in \{10, 30, 50, 70, 90\}$, we generate 2000 DCOPs, each one with $1.5 \cdot n$ constraints whose arity is randomly picked from 2 to 4. We create pseudotrees for DPOP using the DFS-MCN heuristic [9] and DJTs for Action-GDL considering the rearrangement of the DJT produced by mapping γ as explained above. Figure 7 (upper) shows the average savings (in percentage) in communication and computation of Action-GDL with respect to DPOP[3]. Observe that a simple rearrangement of the DJT leads to significant savings in communication and computation costs, which increase as the number of variables grows.

In our second experiment we show that we can help Action-GDL to reduce the maximum degree of parallelism with respect to DPOP. We propose to found such improvement on another rearrangement of the DJT produced by mapping γ. This time we propose to change the root of the DJT. Figure 6(d) illustrates such rearrangement for the DJT in figure 6(b). Observe that changing the root of a DJT never changes either the computation or the communication costs because cliques and separators remain the same. Notice also that we cannot explore such an arrangement in DPOP because changing the root of a pseudotree can lead to a non-valid pseudotree. For instance, choosing x_2 as a root in the pseudotree of figure 6(a) makes it an non-valid pseudotree (because of the dependency between variables x_1 and x_3). Next we measure and compare the MPC, formally defined as $MPC = \max_{P_i \in P} \sum_{C_j \in P_i} d^{|C_j|}$, where P stands for the set of all paths, a path P_i contains all cliques from the i-th clique to the root, and d stands for the variable domain size. To run this experiment we employ the same pseudotrees generated for our first experiment above and we set $d = 2$. We rearrange DJTs for Action-GDL as explained above to select as clique root the one that reduces the MPC the most. Figure 7 (lower) shows our empirical results by depicting the average (in percentage) improvement in MPC[3] that Action-GDL achieves. Observe that the gain in parallelism can be very significant (from 25% to 40% of MCP reduction), and it increases as the number of variables grows.

[3]Percentage assessed as $\frac{(DPOP - ActionGDL)}{DPOP} \cdot 100$

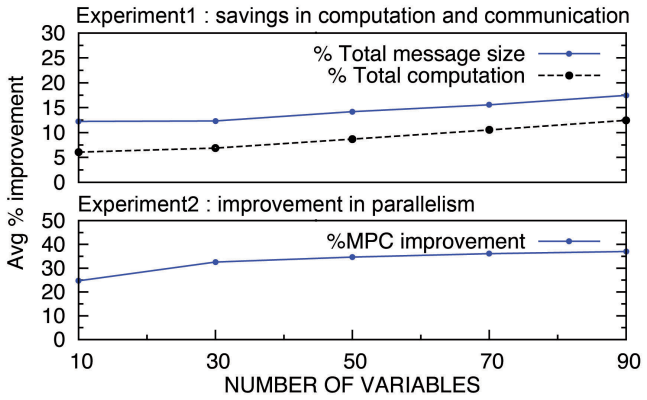


Figure 7: Experimental results

7. CONCLUSIONS AND FUTURE WORK

We made three main contributions in this paper. Firstly, we presented a new algorithm, the so-called Action-GDL, as an extension to GDL [1] to efficiently solve DCOPs. Secondly, we introduced the Distributed Junction Tree Generator (DJTG) algorithm, which allows agents to distributedly compile a distributed junction tree over which Action-GDL can operate. Finally, we show that Action-GDL generalizes DPOP. To do so, we prove that: (1) DPOP is a particular case of Action-GDL; and (2) Action-GDL can exploit distributed junction trees as a more general structure to generate executions that cannot be achieved by DPOP via pseudotrees. Moreover, we provide empirical evidence to show how we can computationally exploit the generality of Action-GDL. Thus, we show that Action-GDL can outperform DPOP in terms of the amount of computation, communication and parallelism of the algorithm solving cost. Finally, we argue that there are also analitical reasons to prefer Action-GDL. Since it is based on GDL, we can benefit from a wealth of theoretical results for GDL over junction trees and other approximate or more general structures such as single-cycle junction graphs [1].

8. REFERENCES

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] J. Atlas and K. Decker. A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *AAMAS*, page 111, 2007.

[3] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, 1987.

[4] R. Dechter and J. Pearl. Tree clustering for constraint networks (research note). *Artif. Intell.*, 38(3):353–366, 1989.

[5] F. V. Jensen and F. Jensen. Optimal junction trees. In *UAI*, pages 360–366, 1994.

[6] J. L. K. Kask, Rina Dechter and A. Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artif. Intell.*, 2005.

[7] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, 2005.

[8] M. A. Paskin, C. Guestrin, and J. McFadden. A robust architecture for distributed inference in sensor networks. In *IPSN*, pages 55–62, 2005.

[9] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, EPFL, Lausanne, 2007.

[10] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[11] M. Vinyals, J.A.Rodriguez-Aguilar, and J. Cerquides. Proving the equivalence of action-gdl and dpop. Technical report, IIIA-CSIC, 2008. Available at `http://www2.iiia.csic.es/~meritxell/publications/TRR200804.pdf`.

Distributed Constraint Optimization for Time-Critical Domains

James Atlas
Computer and Information Sciences
University of Delaware
Newark, DE 19716
atlas@cis.udel.edu

Keith Decker
Computer and Information Sciences
University of Delaware
Newark, DE 19716
decker@cis.udel.edu

ABSTRACT

Distributed Constraint Optimization (DCOP) provides a rich framework for modeling multi-agent coordination problems. Existing problem domains for DCOP focus on small (<100 variables), deterministic domains. In many real-world, time-critical domains, agents are given limited information about the problem environment and are expected to coordinate with limited computation and communication. We present a complete DCOP solution for one such domain, 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. This algorithm 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.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

Distributed Constraint Optimization, Multi-agent Coordination, Task Scheduling

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[8], traffic flow cooperation [5], and event scheduling [7]. 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 compare our new algorithm with existing DCOP algorithms on standard graph coloring problems and in the Coordinators domain. 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 [4]. 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 have a quality accumulation function (QAF) that describes how quality accrues at the task based on the quality of its subtasks 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

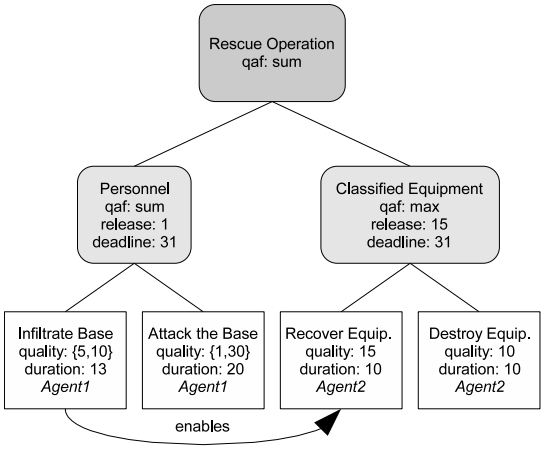


Figure 1: An example C-TÆMS problem instance.

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 [6, 13, 11]. 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 [10]. A Distributed Constraint Optimization Problem with n nodes and m constraints consists of the tuple $< X, D, U >$ where:

- X = $\{x_1,..,x_n\}$ is a set of variables, each one assigned to a unique agent
- D = $\{d_1,..,d_n\}$ is a set of finite domains for each variable
- U = $\{u_1,..,u_m\}$ 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 = $\{u_1,..,u_m\}$ 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 = $\{(u_p^1,\ u_v^1),..,(u_p^t,\ u_v^t)\ \}$ is a distribution of probabilities and values such that $\sum_{r=1}^{t} u_p^r = 1$
- E = $\{e_1,..,e_n\}$ 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 local-search DCOP algorithms sum together constraint valuations and then use a max function to make decisions about the best local variable assignment choice. Sum functions are easily extendable to discrete distributions, but to take the maximum of a distribution an algorithm must apply an evaluation function. In our model evaluation functions in E can be different for each variable, but in practice all variables use the same evaluation function. The place at which a DCOP algorithm takes a utility distribution and evaluates it using the function in order to make a comparison is algorithm specific. The earlier such a measurement is taken, the more information about the distribution is lost. For instance, we could apply the evaluation function immediately to all constraint valuations and lose all information about the distribution.

In practice, most local-search based algorithms provide a point at which the possible values in D_j for variable X_j are tested to see which is the best given known information about the current values of all neighboring variables to X_j. To employ the evaluation function for X_j in a simple algorithm such as MGM[2], all utility distributions for utility functions involving variable X_j are summed together for each possible assignment $a_k \in D_j$. We then choose the maximum utility assignment by applying the evaluation function to these sums and choosing assignment corresponding to the maximum value. This example only looks at immediate neighbor assignments because it is based on MGM, but in general any algorithm that uses sum and max functions to make decisions can be adapted to use utility distributions by summing utility distributions and using the evaluation function to perform the max comparison.

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. An example of this is shown later in Section 5.2.

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 [12]. 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. It allows only sum, min, and max QAFs, and all QAFs in the same problem must be of the same type (no mixing sum with max QAFs, or taking the max over a set of sums). It also only allows enables NLEs and requires deterministic task outcomes, so it cannot handle NLEs that are contingent upon the outcome of a method or method's with non-deterministic quality or duration.

An extension to this mapping added syncsum QAF mappings and provided for multiple QAF types in the same structure without using n-ary constraints[1]. 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 Using Utility Distributions

To illustrate the usefulness of our earlier model of DCOP with Utility Distributions for this domain, lets look at the simple CTÆMS structure in 1. In this example, Agent1 does not know what the quality outcomes of *Infiltrate the Base* (M1) and *Attack the Base* (M2) will be apriori. Agent2 knows the outcomes for *Recover Equip.* (M3) and *Destroy Equip.* (M4), but is dependant on Agent1's choice of execution because of the enables relationship. Based on the deadlines involved, the agents can either execute M1+M3 or M2+M4 (M1+M4 is possible but clearly inferior to M1+M3). How does Agent1 choose whether to execute M1 or M2? Assuming uniform probability distributions, let us consider several evaluation functions:

- *Average Value* - takes the sum of the product of probability and utility. For M1+M3 this is $(0.5*5+0.5*10)+15 = 22.5$ and for M2+M4 this is $(0.5 * 1 + 0.5 * 30) + 10 = 25.5$. M2+M4 is the optimum choice here.
- *Minimum Confidence* - calculates a percentage separator over the set of probabilities. A 50% minimum confidence is the median value. If we took a 25% minimum confidence separator over these distributions we would get M1+M3 as $5 + 15 = 20$ and M2+M4 as $1 + 10 = 11$. Thus M1+M3 is the optimum choice.
- *Quadratic Risk Aversion* - also calculates a value that tries to minimize risk. Calculates a new weighting for the sorted values (from lowest quality to highest) in the discrete distribution using integration over the function $2 * (1 - x)^2$. For M1+M3 it evaluates to $(0.875*5+0.125*10)+15 = 20.625$ and for M2+M4 it is $(0.875*1+0.125*30)+10 = 14.625$. M1+M3 is also the best choice for this function as well.

We see that the average value function does not take any risk into account. The second function is mildly risk averse but doesn't really take into account the whole distribution. The third function is strongly risk averse and does evaluate over the whole distribution. The other thing we see is that the evaluation for risk occurs locally to the distribution. However, the risk for most problems is a global risk because many methods have non-deterministic outcomes. If we had 10 methods with similar distributions to M2, executing all of them significantly reduces the risk of getting a really low total quality ($< 0.1\%$ chance of getting below 30 quality). We could not determine this if we pre-evaluated the distribution locally as it appears to be very risky with a 50% chance of returning only 1 quality. Since DCOP algorithms try to maximize the global solution, incorporating risk evaluation requires propagation of utility distributions through the constraint network so that evaluation functions can operate on combined distributions. We integrate this method of evaluation into the Distributed Neighbor Exchange Algorithm presented later in Section 6.

5.4 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.

5.4.1 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.

5.4.2 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.

5.4.3 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*.

5.5 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.

5.5.1 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.

5.5.2 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.

5.5.3 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 $\max_{k \in C_M} -QM_k$ where C_M is the set of methods involved in the constraint.

5.5.4 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 hinderence 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. DNEA can also be extended for our model of DCOP with Utility Distributions using the techniques presented earlier, and this has been done for our solution to the DARPA Coordinators scenarios in 7.

6.1 Algorithm Phases

Our algorithm is similar in phases to other local value exchange based algorithms, including MGM, SCA, DBA, and max-sum[2, 3]. 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.

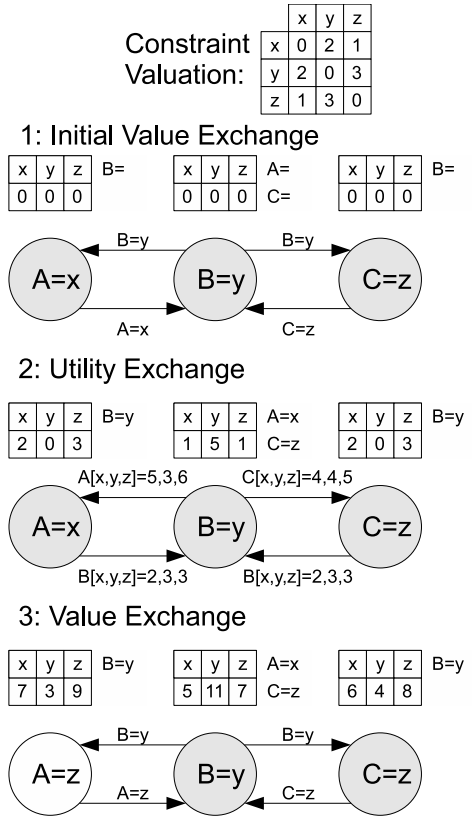


Figure 2: DNEA in action. Given constraint valuations shown at top and a random starting assignment of A=x, B=y, and C=z, DNEA finds the optimal assignment in one round of exchanges. Note how the utility exchange message from B to A in step 2 contains aggregated utility from C.

6.1.1 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.

6.1.2 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. We define N_X as the neighbors of X to be all variables with whom X has a constraint. Maximum computation per variable per phase for DNEA is then $O(|N_X| \cdot |D_X|)$ for value exchange and $O(\sum_{Y \in N_X} |D_Y| \cdot |D_X|)$ for utility exchange. Maximum message size for value messages is $O(1)$ and for utility messages is $O(|D_X|)$. The number of messages per phase is the same as other algorithms as each variable sends one message to each of its neighbors for each phase.

The tradeoff then is that DNEA does more calculation during utility exchange phase than DSA and SCA2, but less than SCA3 as long as the number of neighbors for a variable is less than the number of possible values in that variable's domain, $|N_X| < |D_X|$. From [14] and [2] the calculation per phase for DSA is $O(|N_X| \cdot |D_X|)$, for SCA2 is $O(|N_X| \cdot |D_X| + |D_X| \cdot |D_Y|)$, and for SCA3 is $O(|N_X| \cdot |D_X| + |D_X| \cdot |D_Y| \cdot |D_Z|)$. Per message size for DNEA is larger than DSA which is $O(1)$, similar to DSA2 which is also $O(|D_X|)$, but smaller than SCA3 which is $O(|N_X| + |U_X|)$ where $|U_X|$ is the size of all of the constraints at X sent to the group offerer.

6.3 Comparison with other DCOP algorithms

We tested DNEA against four local search algorithms, DSA and DBA from [14], and SCA2 and SCA3 from [2]. We made one optimization for these algorithms that significantly improved their performance: all variables were allowed to change (or offer a group change) values if the gain was greater than or equal to zero (instead of only greater than). We chose these local search algorithms because they can also scale to C-TÆMS problems containing 1000+ variables.

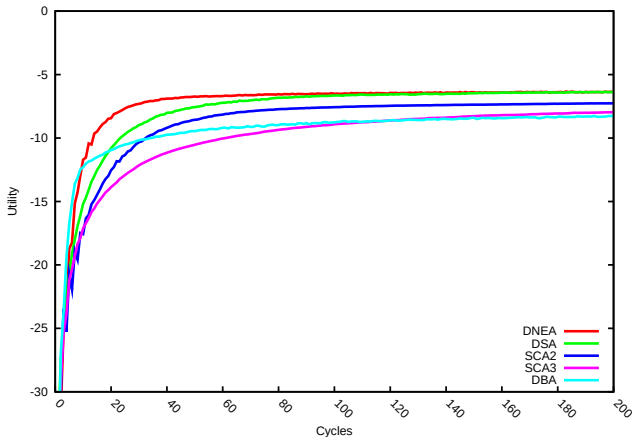


Figure 3: Graph Coloring Problems: 40 variables (200 cycles)

6.3.1 *Comparison for Graph Coloring Problems*

We performed comparisons for standard graph coloring problems that can be found in the DCOP repository at USC [9]. There are 25 problems that contain 40 variables each, with 120 constraints, and 3 possible colors for each variable, with constraint violations for connected variables of the same color worth -1 utility. Each problem was run 100 times for each algorithm with different random number seeds and results are shown as the average global utility at each message passing cycle in Figure 3.

We observe that DNEA reaches a high utility by cycle 20 and the highest utility by cycle 40. This was the goal in the creation of DNEA, to be able to achieve high utility with very few message passing cycles while keeping polynomial time calculation for each cycle. We do not show it here, but eventually SCA2 and SCA3 catch up; however, it takes well over 500 cycles to do so. Additionally, we show that our earlier theoretical comparison of message size and calculation matches our empirical tests; averages are shown in Table 1.

In Table 1 we show two measures taken during each message passing cycle: non-concurrent constraint checks (NCCC) and non-concurrent message transfer size (NCSize). For these metrics we measure the largest number of constraint checks performed and the largest message processed by any agent during each message passing cycle. We then sum over all of the cycles to produce a view of the total non-concurrent constraint checks (NCCC) and total non-concurrent message transfer size (NCSize). We also show total constraint checks (TCC), total number of messages (TMsgs), and total message size (TSize) although these metrics are a simple sum and do not take into account the parallelism involved. SCA2 and SCA3 exhibit lower total constraint checks because roughly 1/2 and 1/3 of all nodes are actively processing each cycle respectively because of how the algorithms create groups of 2 and 3 nodes respectively. However, this also means that inactive nodes are idle and must wait for other nodes to complete processing, so the non-concurrent (sequential) processing is not lower because of this.

6.3.2 *Comparison for Static C-TÆMS*

In addition to the graph coloring problems, we apply our C-TÆMS mapping in a static scenario (the agents are searching for the schedule with best expected utility but no methods are executed and scenario time stays at 0). This scenario was run in two parts, one randomly generating an initial schedule for the agents, and one using an initial schedule generated by an offline deterministic solver. Each scenario was run 10 times using different random seeds. Detailed description of the scenarios ("OptOP5PMix" and "OptBigReMix") are provided in Section 8. Results are shown in Figures 4, 5, 6, and 7.

Algorithm	NCCC	NCSize	TMsgs	TCC	TSize
DNEA	19062	6000	54422	249367	1413775
DSA	9439	3600	48177	194388	867186
SCA2	14532	8409	29200	152182	538559
SCA3	22555	77795	26026	82329	780317
DBA	4445	4800	54009	58857	1133794

Table 1: Graph Coloring Problems: 40 variables (200 cycles)

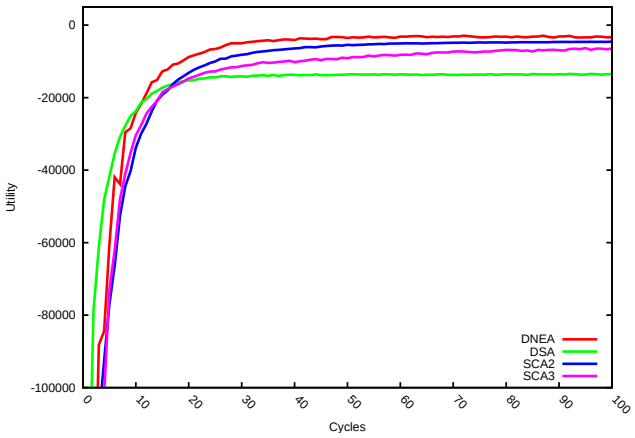


Figure 4: OptOP5PMix: ∼100 variables, random initial schedule

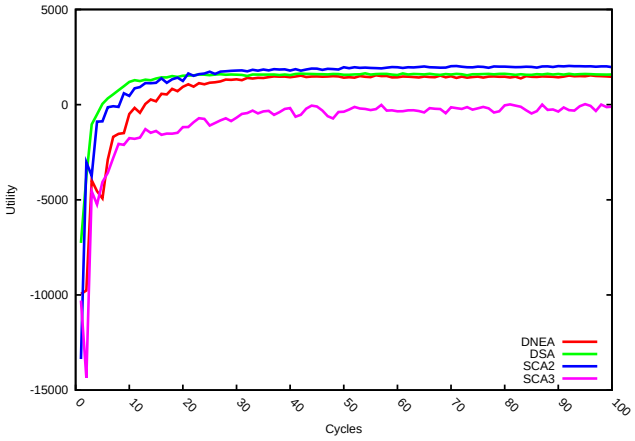


Figure 5: OptOP5PMix: ∼100 variables, given initial schedule

It is clear that for randomly generated schedules, DNEA converges quickly to the schedule with the best estimated utility. We see this again at cycle 20 in both Figures 4 and 6. This result led us to use DNEA as the base algorithm in our Coordinators simulation, as dynamic outcomes during execution may require quick re-scheduling and possible generation of a vastly different schedule. However, for established schedules DSA and SCA2 are able to improve them slightly more than DNEA. In all cases SCA3 requires too many cycles to converge to high quality solutions in this

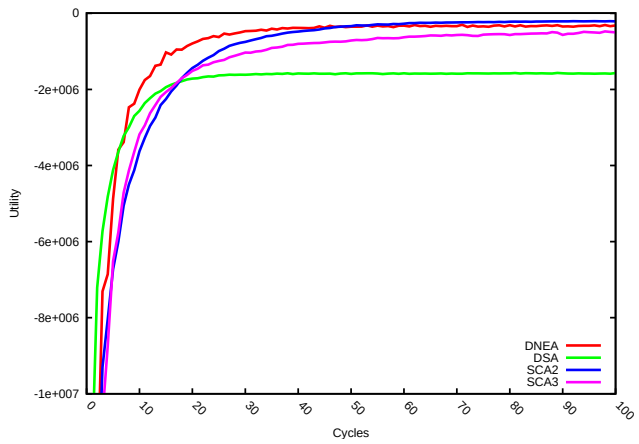


Figure 6: OptBigReMix: ∼1000 variables, random initial schedule

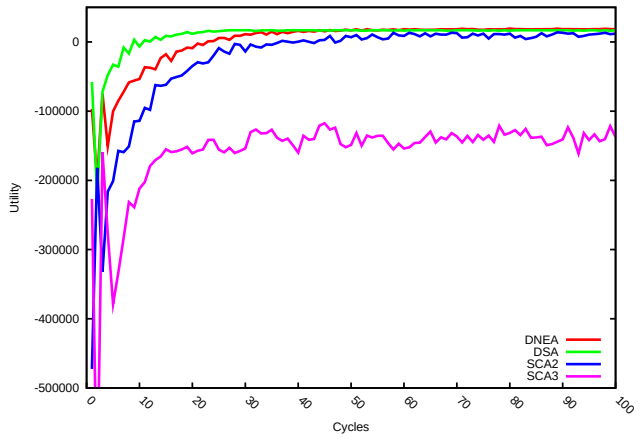


Figure 7: OptBigReMix: ∼1000 variables, given initial schedule

domain. We are working on a multi-phase algorithm that can begin with DNEA techniques and then continue using SCA2 or SCA3.

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 [15], 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.

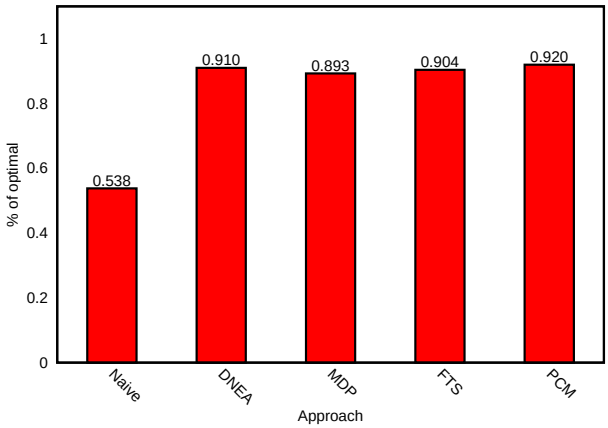


Figure 8: OptOP5PMix: Solution Quality as % of optimal

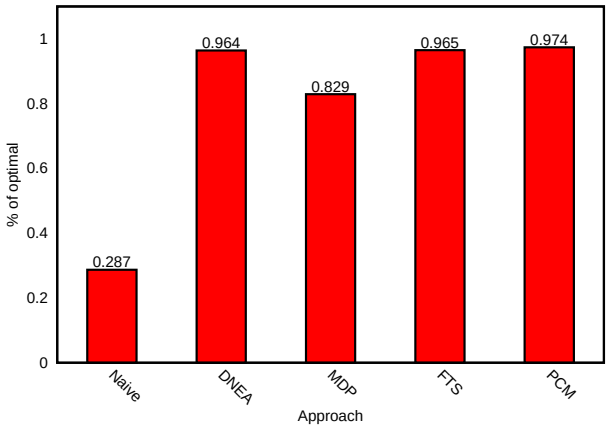


Figure 9: OptBigReMix: Solution Quality as % of optimal

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 8 and Figure 9 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).

9. ANALYSIS

We see clearly that our DCOP approach can achieve much higher performance than the baseline. Our mapping from C-TÆMS 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.

11. REFERENCES

[1] J. Atlas and K. Decker. Task scheduling using constraint optimization with uncertainty. In *AAMAS '07 - Workshop on Coordinating Agents' Plans and Schedules, CAPS*, pages 25–28, 2007.

[2] E. Bowring, J. P. Pearce, C. Portway, M. Jain, and M. Tambe. On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In *AAMAS '08*, pages 607–614, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[3] A. Farinelli, A. Rogers, A. Petcu, and N. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS-08*, 2008.

[4] B. Horling et. al. The TAEMS White Paper, January 1999.

[5] R. Junges and A. L. C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *AAMAS '08*, pages 599–606, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[6] R. T. Maheswaran et. al. Predictability & criticality metrics for coordination in complex environments. In *AAMAS '08*, pages 647–654, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[7] R. T. Maheswaran et. al. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.

[8] P. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. In *AIJ*, pages 149–180, 2005.

[9] J. P. Pearce. USC dcop repository, 2005.

[10] A. Petcu and B. Faltings. Dpop: A scalable method for multiagent constraint optimization. In *IJCAI 05*, pages 266–271, Edinburgh, Scotland, Aug 2005.

[11] S. F. Smith, A. Gallagher, and T. Zimmerman. Distributed management of flexible times schedules. In *AAMAS '07*, pages 1–8, New York, NY, USA, 2007. ACM.

[12] E. Sultanik, P. J. Modi, and W. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, 2007.

[13] S. Witwicki and E. Durfee. Commitment-driven distributed joint policy search. In *AAMAS '07*, pages 1–8, New York, NY, USA, 2007. ACM.

[14] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS 03*, pages 185–192, New York, NY, USA, 2003. ACM Press.

[15] R. Zivan. Anytime local search for distributed constraint optimization. In *AAMAS 2008 - DCR Workshop (Distributed Constraint Reasoning)*, 2008.

Towards Efficient Coordination in Open MAS using Graphical Utility Models

Nicolas Stefanovitch
Amal El-Fallah Seghrouchni
Frédéric Peschanski

Université Pierre et Marie Curie - Paris 6
UMR 7606, LIP6
4 Place Jussieu, Paris,
F-75005 France

ABSTRACT

This paper presents a prospective work on the problem of efficient agent coordination in open and decentralised multi-agent systems (MAS). We use a graphical utility model, the General Additive Independence networks (GAI-net), as the basic tool for coordination inside MAS. We extend this formalism and propose two algorithms in order to take into account inherent MAS operational characteristics. The first algorithm is able to deal efficiently with a restricted notion of openness. It provides an optimal solution but requires a central coordinator. The second algorithm we propose takes moreover into account the decentralised nature of MAS. It is an approximate algorithm with performance guarantee.

Keywords

graphical models, junction tree, social welfare, multi-agent systems, openness, decentralisation

1. INTRODUCTION

Multiagent systems are a versatile tool, mainly a way of conceiving the organization of complex systems as multiple self-behaving local entities (agents). The purpose of such a system is to solve a given problem. The originality of the MAS approach is to distribute the data, the computing facilities and the system's goals within the agents. Each agent has a set of goals he is due to accomplish by taking his own decision. However because agents are part of the same system, they are tied together through a common environment. The effects of each agents' decisions might then affect the outcome of other agents' decisions. Agents, either cooperative or selfish, therefore have to coordinate their decisions in order to maximise their own satisfaction. Such a satisfaction can be described using utility functions. A common way of describing the optimality of a joint action is to measure its social welfare, which is the sum of the utility of all the agents. Therefore coordination between benevolent agents should aim at maximising this function. However the direct maximisation of this function implies a costly computation exponential in the size of all the decision variables present in the MAS. In addition such an approach fails to exploit the local independencies existing between agent's utility function. Graphical utility models, such as the GAI-networks [1] framework propose an efficient solution to this kind of maximisation problems. In this paper we are interested in operational issues when using such a framework on top of a MAS. As a matter of fact, the decentralised and open nature of MAS needs to be taken specifically into account so as to design efficient coordination protocols for real-world MAS.

Graphical models were first introduced with Bayesian networks so as to make efficient computations over probability distributions. More precisely graphical models are used to store multi-attribute functions into a compact form exponentially smaller than their flat representations. Such a compact representation is possible only if a multi-attribute function is decomposable following a certain independence relation. A graphical structure represents the interaction between the parameters of the function. A set of sub-functions is attached to the nodes of this graph. Inference is performed using a message passing scheme, known as belief propagation. This algorithm is exact only for acyclic graphs. A secondary graphical structure, the junction graph, must be build in order to perform exact computation when the graph is cyclic. This graph is tree structured and it is obtained by applying the variable elimination algorithm over the primary graphical structure. Graphical structure are generalised by the factor graphs and the sum-product algorithm [5].

The link between MAS and graphical models is particularly attractive as graphical models capture the local structure of a MAS and proposes moreover an inherently distributed inference algorithm to solve inference tasks. Several works have already explored this link, and can be decomposed roughly into three categories: Distributed Constraint Optimisation Problem (DCOP) [12] for collaborative agents, graphical games for competitive agents, and distributed probabilistic inference. The domain of application is wide as it concerns almost any aspect of multi-agent interaction, and ranges in wireless sensor network (WSN), multiagent scheduling or teamwork planning for naming few of them.

Graphical games [6], explores a non-cooperative agent setting and is concerned with the computation of Nash equilibria, graphical models are used purely for their computational advantages. They are solved through the use of a constraint satisfaction solver, using variable elimination or backtracking techniques. While we are concerned with

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

a distinct problem, the mechanisms are nevertheless similar. Notably [4] sketches an approach to compute approximated Nash equilibria over a junction tree with a principle similar to the one we use in the approximated algorithm we present in this paper. Multi-agent Influence Diagrams (MAID) [3], are graphically standard influence diagram representations where decisions and reward nodes correspond to distinct agents. The problem there is finding Nash equilibria over expected utility of the agents using backward induction, and is therefore merely related to our work. Variable elimination have been used in [2] to solve approximated factored MDP for MAS planning, however the approach is totally centralised. Finally, in [7] the authors consider the problem of robust probabilistic inference in open WSN, similar to our concerns. Their anytime approach is very interesting, however the inference mechanism is specific to Bayesian networks, while we are interested in utility functions.

The DCOP domain is highly linked to our work as GAI-networks maximisation is in fact a constraint optimisation problem. The work more closely matching ours is the DPOP protocol [9]; which implicitly builds a junction graph when building a search tree over the interaction graph. However, being able to separate the selection of an order on the variables and the selection of a root, which is not possible in DPOP, is an important concern in our work so as to bound the total running time. In [11] the authors consider openness only in the cardinality of the variables. In [10] the authors propose ODPOP an extension of DPOP dealing with the general case of openness: both in value and in structure of the graphical model. Our work focuses on re-using previously made computations so as to improve the reactivity. In the case of a single modification the ODPOP framework, while being also able to re-use past computations fails to do it optimally as we explain how. In the case of a multiple modifications, the ODPOP framework proposes to create as much centralized instances of the inference algorithm as there is variables in the system. This approach is burdensome as the transmitted messages are of exponential size, and moreover it is totally centralised. Our approximate approach on the contrary makes parsimonious use of resources and takes into account the decentralisation of the MAS.

This paper is organised as follows: in section 2 we present an intuitive connection between GAI-nets and MAS coordination, and pose the problem settings. In section 3 we present a first direct extension of the GAI-net framework allowing exact and efficient inference in open MAS, under a restricted notion of openness. In section 4 we present an approximated algorithm with performance guarantee, which addresses the issue of decentralisation, allowing faster coordination for certain classes of MAS, and finally in section 5 we conclude.

2. DISTRIBUTED GAI-NET INFERENCE

2.1 GAI-net

GAI functions can be efficiently computed using a graphical model called a GAI-network [1]. This model proceeds by building a junction graph from the dependencies expressed in a set of sub-utility functions. The requests are processed along this graph, using an instantiation of the sum-product algorithm for utility functions. While a junction graph has a forest structure, we will consider for simplicity purpose that the junction graphs we deal with are tree-structured. In this work we are only interested in the maximisation request, which computes a maximal value for the set of utility functions U from which a GAI-net G has been built.

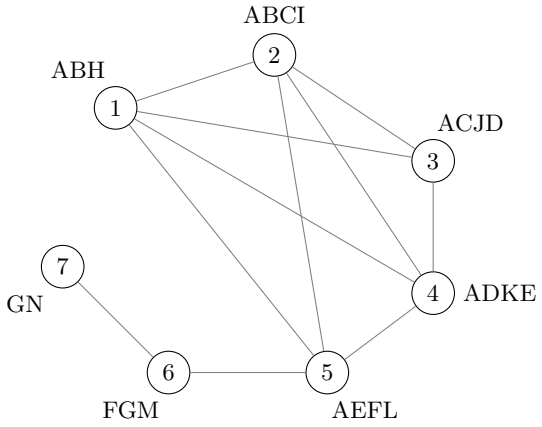


Figure 1: Dependencies between MAS's agents

The multi-agent systems we consider are constituted of a set A of agents, each of them possessing an utility function $u_a \in U$ describing his preferences. The domain of an utility function u is $dom(u)$, which is a subset of the set $\mathcal{X}$ of all the decisions variables of the system. Each agent has the goal of maximising his own utility. However as agents share a common environment, their decisions are tied. Such an interaction is represented between two agents by a non null intersection of the domains of their utility functions. Therefore the agents, need to coordinate so as to ensure a good performance. We consider in this work benevolent agent cooperating so as to maximise the welfare utility $w(x) = \sum_{a \in A} u_a(x)$ of the system. In a mathematical setting, such a problem is precisely the one that is efficiently addressed by GAI-networks. As the GAI-net inference algorithm is inherently distributed, the setting where one agent represents a clique of the junction graph is already naturally addressed by this algorithm.

However even assuming the unlikely hypotheses of one agent per clique, this framework fails to be plainly satisfactory from a MAS perspective. The reason is that MAS are more than just distributed systems, MAS are open and decentralised distributed systems. These two characteristics raise serious operational issues that are not taken into account within the mathematical formulations.

2.2 GAI-net inference over a MAS

Before focussing our attention on those issues, we will first describe how the standard GAI-net inference algorithm performs on a MAS. We consider a MAS A composed of seven agents whose respective preferences ranges over: ABH, ABCI, ACJD, ADKE, AEFL, FGM, GN. This example will be used throughout the paper, it is depicted in Figure 1, where vertices represent agents and edges link any two agents sharing at least a common variable.

So as to give an intuitive MAS feeling of the procedure, we will first start by considering how the agents would interact without a GAI-net. As the goal is to maximise the welfare utility, the agents could collect in one of them all the utility functions of the system. Such an approach is both undesirable from a MAS and from a computational perspective. Firstly because this approach is totally centralised, a failure

in the collecting agent cripples the whole system. Secondly, from a computational perspective, the centralising agent not yet knowing about GAI-nets would merely construct and perform an exhaustive search exponential in $|\mathcal{X}|$. This approach fails to exploit the existing additive independences between the agents' utilities. Now let us consider a second better coordination scheme: each agent propagates his utility tables only to agents that share decision variables with them. Upon receiving a message an agent creates a new message containing all the information that he has received but not yet transmitted, be it his own utility table or another messages. While each agent shares their information with random neighbors, let us consider the case that an agent a_X happens to possesses a set T of utility tables containing all the utility tables of A depending on a variable $X \in \mathcal{X}$. This agent is then already able to realise a maximisation of U over X: for every value of X he selects a corresponding maximal value of T over the domain $D = dom(T) - X$. As a_X does not know the optimal values of D he cannot already give to X its optimal value. Such an operation is the *collect* procedure in the GAI-net framework, and it results in the creation of a new utility table over D. Such an utility table summarises the utility information of various agents into a new utility table that can also be propagated, such tables are referred to *messages* or *separators*. The agent a_X has played the role of a clique of label $dom(T)$, and by creating a new message of domain D has implicitly binded together all the variables of D, so therefore every agent receiving this table would have D as a subdomain.

This process of eliminating one after another each variable is known as *variable elimination*. While this approach proceeds in a clear multi-agent way, its efficiency is still debatable nevertheless. As a matter of fact the computational complexity of this second approach is exponential in the size of the label of the biggest clique, which without care could be as big as $|\mathcal{X}|$. The size of the cliques dynamically created by the elimination procedure depends on the elimination order of the variables. Finding an optimal elimination order, in the sense of the minimal cardinality of the biggest clique, is a NP-Complete problem. However the polynomial heuristic of selecting the variable in the order of their increasing cardinality builds near optimal cliques. In GAI-net this is done through the Markov graph of U, the Markov graph of our example MAS is given in Figure 2. This graph is dynamically updated: each time a variable X is eliminated, the set D of its neighbors is updated. When the elimination is completed, each clique is linked to the clique created during the elimination of the least recent variable in its label - corresponding to the creation of a message. By doing so, the structure we obtain is an elimination tree. It does not take into account that multiple variables may be eliminated at the same time. However, merging the subsuming cliques into subsumed cliques ensures that no unnecessary message will be computed. The resulting structure is the junction tree of a near-optimal GAI-net of U.

Let us get back to our MAS. While we started with the utility stored in the agents, we dynamically created cliques containing utility tables not necessarily originating from an utility of U, but from messages created by the agents. We have proceeded in such a way so as to give an intuitive connection between GAI-nets and MAS: from a multi-agent perspective a GAI-net is a dynamically generated near-optimal coordination protocol for the maximisation of the social welfare of the system. In practice no actual computation is

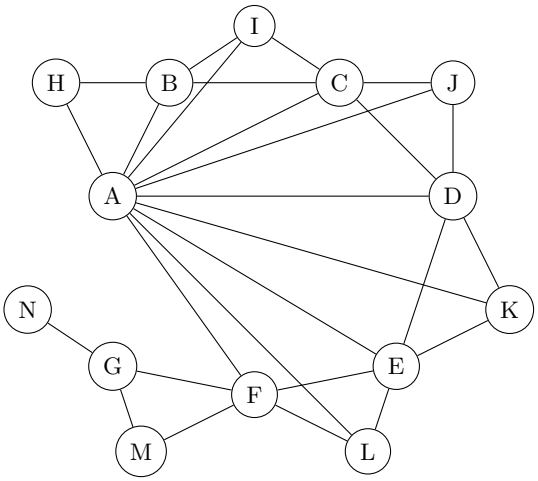


Figure 2: Markov graph of MAS's variables

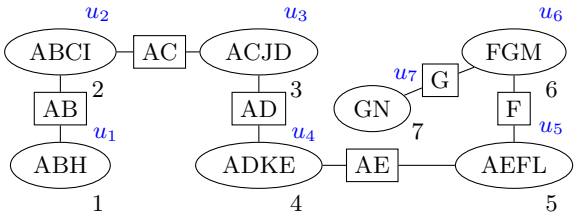


Figure 3: Junction tree of the MAS

made during the building of this structure. Computations are made after the following initialisation phase. Each table $u \in U$ is allocated to one of the clique c of G such that $dom(u) \subseteq dom(c)$. A clique is a role that can be played by any agent. From a DAI perspective, this is just a parallelization problem, but from a multi-agent perspective there is no special-purpose computational agent, and a reasonable compromise between the agent's privacy and necessary information propagation is to attribute the role of a clique to one of the agents whose utility table or messages is associated with. It is heuristically desirable so as to minimise bandwidth usage and completion time that this agent be the one with the biggest clique cardinality. The last point so as to have a fully-functioning GAI-net working on top of a MAS, is to give an arbitrary direction to the separators of the junction tree. By choosing a root clique the structure becomes ordered, and knowing the utility attribution to cliques, and the clique attribution to agents, a near-optimal structure of interaction between the agents is automatically derived. We present the resulting junction tree for our MAS example in Figure 3, and the derived interaction structure when choosing arbitrarily the clique ABCI as the root in Figure 4.

The GAI-net maximisation algorithm proceeds in the following way: the root calls the *collect* procedure over each of his neighbors. Those neighbors, prior to maximising their local utilities and set of received messages, recursively call the *collect* procedure over their neighbors except the node that called the procedure over them. Once the root node received

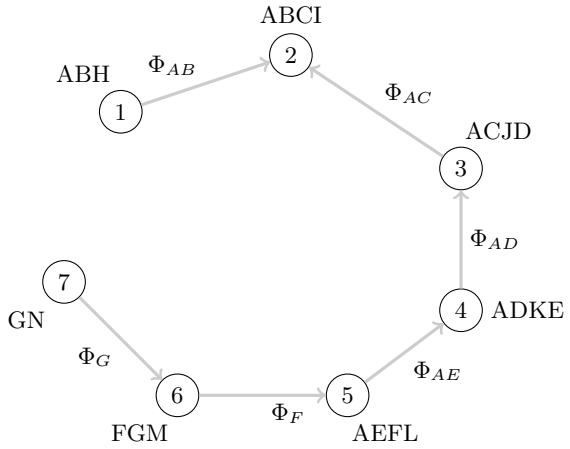


Figure 4: Message exchange inside the MAS during inference

all its messages, it optimises its local values, and starts a procedure called *distribute*: each node upon receiving a partial assignment of optimal values of $\mathcal{X}$ completes this assignment with their optimal local values and transmits it recursively to their neighbors. Optimising the partial assignment computed at the root level is globally optimal because the message received by the root contains a summary of all the information in the system. Optimising partial assignment at the node levels of non clique node is globally optimal because the way the junction tree is constructed enforces the *running intersection* property. This property states that if a variable appears in two cliques, it also appears in all the cliques on the path joining them. A variable assignment can therefore be chosen only once, which prohibits inconsistent assignments. We refer to [1] for the formal description of the model and algorithms.

2.3 MAS specificities

The GAI-net inference mechanism we have just presented is very interesting for representing MAS coordination as it captures the locality of interactions and is naturally distributed. However it fails being fully satisfactory for MAS in two regards: openness an decentralisation. We introduce in this subsection these two issues.

2.3.1 Decentralisation

Decentralisation can be understood under two acceptations. In distributed systems, this means that there is no single point of failure. Such a point of failure can be a centralising agent with global information about the system, or an agent playing an unique role in the system. In this setting replication of critical roles is a way to ensure the robustness of the system. While this is also a matter, decentralisation in MAS encompasses a distinct notion which deals with their inner structure. Decentralization in MAS means that every agent has local knowledge and is involved only in local interactions, therefore even a robust centralising agent is not an acceptable solution. We are in this work mainly interested with the MAS aspect of decentralization. Making robust the unique clique roles by replicating them, while necessary, do not fall into the scope of this paper. In the remainder of this paper we will be concerned only with the properties of the inference procedure. Let us just say that the junction tree can be constructed in such a decentralised way. Moreover, the knowledge of MAS's Markov graph can be constructed in an incremental and local way, with one agent responsible for each variable.

2.3.2 Openness

Openness signifies that the system is dynamic and inherently uncertain. The system can therefore change unexpectedly. From an agent perspective this means that agents can enter or leave the system at any moment. Such dynamic changes have an impact over the task performed by the agents in the system. As agents carry with them their preference utility, this means that the mathematical form of the welfare function is constantly changing. As a consequence, previous agreements may fail to be optimal, and moreover the coordination structure represented by the GAI-net of the system may become clearly non-optimal. Those two issues tied to openness are of critical importance, and are totally overlooked in the standard GAI-net framework. We will in the remainder of this paper present two solutions addressing the first of the two issues: *ie.* dealing with changing clique values. We thus make the restrictive assumptions that agents entering and leaving the MAS do not change MAS's Markov graph, ensuring the structure of the junction tree remains unaltered. We make this assumption because even slight changes in Markov graph or the elimination order may lead to almost unrelated junction tree. Such a general problem is beyond the scope of this paper. Dealing with openness in the value of the sub-functions already poses problems which are, to the best of our knowledge, unaddressed. This paper is a first and necessary step towards the general case of openness.

3. EXACT METHOD

We present here an extension of the standard GAI-net inference algorithm which is able to deal with simple changes in the set U. One way to address the problem is to restart the inference from the beginning. However such an approach is not wise as only a slight change has happened: we would rather try to exploit as much as possible the already performed computations. Let us see how this is possible in an introductory example.

Consider the interaction structure that we have described in the last section which is depicted in Figure 4. Let us suppose that agent 6 revises his preferences or that a new agent whose utility's domain is a subset of FGM enters the system and that his utility is attributed to clique FGM. Such a modification has no impact on the message computed by agent 7. However, agent 6 must recompute its message to clique AEFL on agent 5. Once agent 5 receives this new message, he therefore also has to recompute his own message. This chain of recomputations propagates through the junction tree until the root is reached, as the root is the only clique which does not compute messages. The root then computes its optimal local values and starts a distribution phase which has to reach every clique. Fig. 5 presents how the communication structure is affected, recomputed cliques and messages are underlined.

The computational complexity of this procedure is exponential in the size of the biggest clique on the path p between the altered clique and the root. From a computational complexity point of view it is therefore interesting to minimise the length of p so as to minimise the chances of encountering

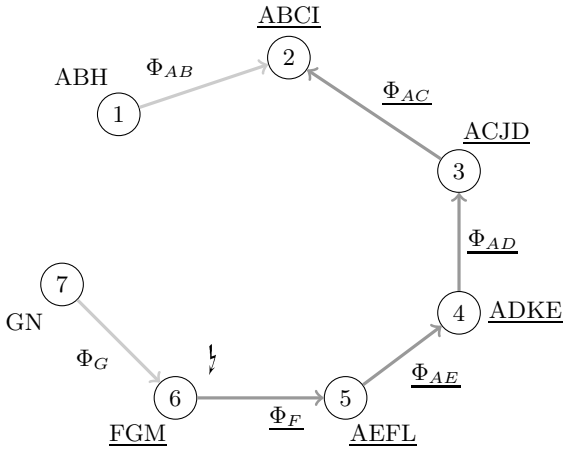


Figure 5: Message exchange inside the MAS when modification occurs

the clique with maximal size of G. From an agent perspective, computational complexity is not as important as the waiting time before a new decision can be made. This time is proportional to $|p|$.

As any clique may be chosen to start the inference, the worst case is therefore a path of length of the diameter δ_G of G. If we consider that a modification can affect all the cliques with equal probability, then the average time is proportional to $\delta_G/2$. Changing the root position has therefore not impact on this mean time, but has however an effect on the maximal time. By placing the root node in the middle of the longest path of G we can ensure that the maximal waiting time would be always inferior or equal to $\delta_G/2$. The selection of the root is an irrelevant topic in the standard GAI-net framework that has an impact in a MAS setting. As a GAI-net is tree-structured, such selection can be performed in $O(n)$ [8]. Inference algorithm corresponds with few modifications to algorithms 3 and 4 given in the next section.

We have shown how it is possible to minimise the number of computations in order for the agents to re-coordinate efficiently when a modification occurs in a clique. Moreover multiple such modifications can occur concurrently in various cliques. However this solution still fails to be satisfactory as the clique that has the root role centralise the inference procedure. While the clique role may be made redundant or transferable to another agent in case of failure, there is still the concern that wherever a modification occurs in the MAS, a collect phase has to be propagated throughout all the network. We would rather like to define a procedure which is able to take into account the local properties present in the MAS, and turn back to computations on the global function only when necessary.

By examining what happened in the above example, we can see that the root node acted as barrier blocking the modifications appearing in one part of the junction tree to expand further. An intuitive idea would therefore be to multiply the number of root nodes on the junction tree. This would first reduce the length of p, and moreover more closely match the locality of MAS's structure. We present in the next section how we concretise this idea. The solution we propose is an approximate algorithm with performance guarantee.

4. APPROXIMATE METHOD

4.1 General description

Before describing our method, let us start by describing what would be a first bad naive approach. Let us consider a GAI-net with n roots. A simple way of dealing with it is having root nodes ignore each other. Such an approach is equivalent to building n distinct GAI-net on top of the same agents, which is computationally burdensome as the number of tables and computations would be multiplied by n. While being clearly inefficient, another more serious problem lies within: if multiple modifications occur concurrently, each root node would optimise locally its variables before the other modifications reach him and propagates assignment values potentially incoherent with the ones propagated by the other nodes. The resulting performance would be sub-optimal and unbounded.

Our approach consists in segmenting G in various partially independent subgraphs. This is done by selecting a certain number of nodes R of G that will play the role of local-roots. This set of root nodes is selected so as to ensure the property that every non root node has a maximal distance of K to at least one node in R. The set of root nodes performs a local collect over the nodes they are connected with. Once this collect is performed, a new GAI-net G_R is constructed from the information collected by the local roots. By optimising G_R the roots coordinate on a common assignment x_R^* over the set $X_R = \cup_{u:cl(u)\in R} dom(u)$ their variables. Once this agreement has been made, the roots propagate the information about this partial instantiation over the subgraphs they are connected with. This coordination phase is necessary so as to ensure that the propagated assignments are coherent, as each subgraph may be connected to multiple roots. The instantiation x_R^* in G induces a new GAI-net G' constituted of a set $S_{G'}$ of unconnected induced subgraphs. Each of those subgraphs can therefore be optimised independently and locally.

4.2 Algorithm

4.2.1 Initialisation

The construction of the junction tree G is the same as the one presented in section 2. The only difference here lies in the selection of the set of roots R. We have to build R in a way ensuring that any other node of G is distant of at most K to at least one root. So as to ensure this property, we propose the following heuristic, formalised in alg. 1 and alg. 2. This procedure is a simple search algorithm that starting from an arbitrary clique r_i labels any clique at a distance of $K+1$ from r_i as a root node and any clique at a distance of K as a cut node. Cut nodes artificially seperate the junction tree into sub trees: all the nodes comprised between a root r an its cut nodes (included) are attributed to r, and therefore would respond to collect request only originating from r. This is of importance so as not to count several times the information of the non-root nodes at the level of G_R. The results would be sub-optimal for the welfare function and without performance guarantee. We take the opposite stand: instead of counting more information in the function w' optimised by G_R, we count less information in such a way that w' is as close as possible to w by inferior values. This labelling procedure is heuristical, as the number of roots and

cuts is clearly not minimal with respect to the desired property, *ie.* that each node of the graph is distant from at least K from a node in R. In fact the labelling procedure labels nodes distant from $K+1$ and not $2K+1$ in order to ensure that this property will be satisfied. This problem is highly related to problem of the p-medians, which is NP-complete for the general case and which can be solved in $0(p \cdot n^2)$ for trees [13]. The p-median is the problem of finding a set of fixed size such that the distance from every other node to this set is minimised. In our setting the size of R is not fixed, but the maximal distance is. Therefore the minimal size of R such that the distance constraint is verified can be done by launching the p-medians algorithm with incremental values of p. However finding efficiently such a set, is beyond the scope of this paper which aims at presenting the basic principles behind the multi-root approach.

Algorithm 1 labelRoot(node n, node caller)

```
1: n.isRoot ← True
2: n.caller ← caller
3: n.nextRoots ← {}
4: n.elimVars ← dom(n)
5: for all c : c ∈ Γ_G(n), c ≠ caller do
6:    labelCountdown(c,n,n,K+1)
7: end for
```

Algorithm 2 labelCountdown(node n, node caller, node localRoot, int value)

```
1: if value = 0 then
2:    labelRoot(n,caller)
3: else if value = 1 then
4:    n.isCut ← True
5:    n.cutVars ← dom(n)
6:    for all c : c ∈ Γ_G(n), c ≠ caller do
7:       n.cutVars ← n.cutVars - (dom(c) - dom(localRoot))
8:    end for
9: else
10:   n.elimVars ← dom(n) - dom(n.caller)
11:   for all c : c ∈ Γ_G(n), c ≠ caller do
12:      labelCountdown(c,n,localRoot,value-1)
13:   end for
14: end if
```

We continue presenting the idea of our algorithm on our previous MAS example. First let us consider that the GAI-net have been built and that the labelling procedure starts for K=2 with clique ABCI as r_i. In this way ADKE will be selected as a cut-node, and AEFL as an another root. The initialisation algorithm then stops and the edges are oriented towards their respective roots. The result is shown in Figure 6. Messages and cliques are written with variables in parenthesis. This indicates what actual utility tables would be computed if we wanted the multi-root approach to give exact optimal results. As it appears on the figure, this approach augments some cliques and messages' sizes, and even increase the size of two maximal cliques. This is a normal behaviour as proceeding in such a way is the same as using a non-optimal elimination order: the running intersection property is conserved but at the possible expend of an exponential increase in the size of the cliques. We take the other alternative: we keep the cliques at their optimal size, but the variables belonging to the intersection of a cut-node

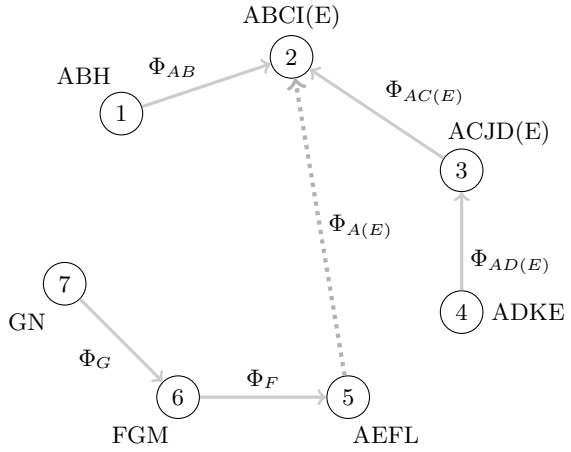


Figure 6: Derived MAS interaction in a multi-root setting

and its root neighbors are removed from the domain of the cut node. Cliques and messages' sizes therefore remains unchanged. However the way variables are removed from the domain of cut node must be done in a principled way.

4.2.2 Approximation

Taking the max over the eliminated variables is not satisfactory as the assignment of variables agreed by the roots may imply selecting a value in a cut node inferior to what was expected, resulting in a lower performance. This is precisely the reason why after the optimisation of G_R we optimise conditionally on x_R^* the whole subgraphs of $S_{G'}$ and not only the cut nodes of G. In order to bound the performance of the algorithm we will therefore take the min of the eliminated variable, the value computed by G_R is thus a lower bound of w.

The approximation is done at the level of the cut nodes of C. A cut node c is adjacent to exactly one another non root node n which is connected to the root r_c on which depends c. The cut node c is also adjacent to zero or multiple roots $R_c \subset R$. The node c will have to modify its utility table u into a new table u' in such a way that all the variables it shares with root nodes are not present in u'. However, when r_c and c share common variables, these can be left in the domain of u' even if they appear in the domain of a root in R_c. The reason is that by removing variables from $dom(u')$ we want to prevent conflicting variable assignments to occur, but when such variables are present in r_c their values will be negotiated during the coordination phase in G_R. The variables to be removed from c's domain are then given by: $dom(c) \cap (dom(R_C) - dom(r_c))$. On the MAS example this signifies that while AEFL$\cap$ ADKE = AE, only the variable E must be removed from the cut node ADKE.

4.2.3 Inference

The coordination at the level of G_R is made using the exact algorithm presented in section 3. After that coordination, all the cliques of G instantiate their utility tables according to the values of x_R^*, resulting in a partition of G in the set $S_{G'}$ of induced subgraphs. The resulting induced cliques are then used to build one new GAI-net per subgraph which can be solved independently of the others.

Algorithm 3 alterClique(node n, set of tables U_{in}, set of tables U_{out})

1: n.localU ← n.localU - $U_{out} \cup U_{out}$
2: **if** n.isCut **then**
3: n.localU ← $min_{n.cutVars}$ n.localU
4: **end if**
5: n.mess ← $max_{n.elimVars}$ n.localU + n.collectedU
6: n.locOpt ← $argmax_{n.elimVars}$ n.localU + n.collectedU
7: inverseCollect(n,n.caller,n.mess)

Algorithm 4 inverseCollect(node n, node caller, table u)

1: n.localU ← removeBySender(n.localU,u.sender) ∪ u
2: n.mess ← $max_{n.elimVars}$ n.localU + n.collectedU
3: n.locOpt ← $argmax_{n.elimVars}$ n.localU + n.collectedU
4: **if** n.isRoot **then**
5: {begin new coordination phase with the other roots}
6: **else**
7: inverseCollect(n,n.caller,n.mess)
8: **end if**

This instantiation results in a decrease of the size of some cliques of $S_{G'}$. Let us consider a clique c which depends on n variables, p of which are present in X_R. So as to compute u'_c a computation exponential in $(n-p)$ must be performed and a table of the same size must be stored. Here a trade-off complexity for memory can be implemented: caching a number exponential in p of tables of size exponential in $(n-p)$ resulting in a total memory consumption per clique of two times the original one. Such a choice depends on the frequency of the changes compared to the time to recompute all the sub-tables. G' is then optimised, and the union of x^*_R and $x^*_{\bar{R}}$ gives the optimal value x^{MR} of the approximate method.

In our example, the set R is constituted of ABCI and AEFL. As these cliques share a common variable, they are connected in G_R, therefore constituting a tree. So as to coordinate one root of R, ABCI, is chosen as the root of G_R; AEFL therefore sends its message to ABCI. The message exchanged during the roots' coordination is depicted with a dashed arrow in Figure 6.

4.3 Bounds

4.3.1 Performance

Mathematically our approach is equivalent to computing first the optimal solution $x^*_{G'}$ of G', then considering the instantiation x^*_R, restriction of $x^*_{G'}$ over X_R, to optimise G conditionally on this instantiation:

$$x^{MR} = \{argmax_{x_{\bar{R}} \in \mathcal{X}-X_R} G(x_{\bar{R}}/x^*_R)\} \cup x^*_R$$

A lower bound of $w(x^{MR})$ is $w'(x^*_R)$. Let $cl(u)$ be the function that returns the clique to which an agent's preference u has been attributed. We have $w'(x) = \sum_{u:u\in U, cl(u)\notin C} u(x) + \sum_{u:u\in U, cl(u)\in C} min_{x_c = u.cutVars} u(x/x_c)$

An easily computable lower bound of $w'(x^*_R)$, noting M the maximal utility in a clique and x^* the optimal solution of G, is $w(x^*) - |C|M$. We can thus give the following performance guarantee to the solution computed by the approximate algorithm:

$$w(x^{MR}) \geq w(x^*) - |C|M$$

The quality of such an approximation depends on M and on $|C|$. M is a factor we can not control, even by assuming that the maximal utility of an agent is bounded, because the total number of agent is not bounded. The factor $|C|$ can be controlled indirectly by the parameter K. Finding the set C of minimum cardinality is therefore heuristically desirable, but without assumptions about agent's utility location this can not by itself ensure that $|u : u \in U, cl(u) \in C|$ will be minimised. The size of this set can however be reduced if after the labelling procedure the attribution of U to cliques avoids the ones in C.

This bound is tighter than the one achieved with the naive approach of cutting the dependencies of the cut-nodes from all the utilities owning those dependencies and then retriangulating the graph, which behaves very poorly. We here only cut a minimal number of dependencies so as to preserve the running intersection property on G'.

4.3.2 Complexity

While the theoretical complexity of the inference remains unchanged, the total running time is changed with respect to the exact method presented above. The total number of recomputed cliques on the path between a modification and a root is bounded by K. Roots coordination implies an additional path of size inferior or equal to $\delta_{G_R}/2$, where δ_{G_R} is the diameter of G_R. This diameter is upper bounded by δ_G/K. Once the roots have coordinated, the length of the path to the root of the GAI-net of the induced subgraphs is inferior or equal to K. Therefore an upper bound of the total running time of our algorithm is $2K + \delta_G/(2K) + 1$, where 1 stands for the computation of the induced sub-cliques. This total running time is what would occur on a parallel implementation with as much processors as cliques. While this is not the case in our settings, this is however not such a problem as it also concerns the exact method, which has a total running time proportional to $\delta_G/2$. However, while the total running time is shortened for carefully chosen values of K, the approximate algorithm performs more computations. More precisely any induced subgraph of $S_{G'}$ which is connected in G to a root that has changed in value must perform recomputations. This implies for any such subgraph s a minimum of $K+1$ computations exponential in the size of the biggest clique in s. In the worst case, when every clique of G depends on a common variable, every clique of every subgraph must make at least one computation exponential in the size of their domains. The exact method presented above, performs a maximum of $\delta_G/2$ such computations whatever the affected clique is.

4.4 Discussion

The total running time of the approximate inference algorithm is guaranteed to be shorter than the one of the exact approach when $2K + \delta_G/(2K) + 1 < \delta_G/2$, which can be rewritten as $(4K^2 + 2K)/(K-1) < \delta_G$. Given δ_G the value of K minimising the total running time of the approximate algorithm is obtained by solving the hyperbolic equation $2x + \delta_G/(2x) + 1 = 0$ which in the case of $\delta_G = 100$ implies $K = 5$. For this value, the approximate algorithm runs at least 10 times faster than the exact approach. Such a behaviour may be desirable in situations where quick response is needed.

However the total running time is not the sole parameter to be considered, and the performance is also a matter. The lower bound we have given for the performance of the

approximate algorithm is $w(x^*) - |C|M$, where x^* is the optimal solution, $|C|$ the number of performed cuts, and M the maximum value inside a clique. While without additional restrictive assumptions we cannot quantify M, $|C|$ is indirectly affected by K. A worst case scenario for the performance is a complete tree. Let us consider a d-ary tree of diameter δ, the number of cuts performed is then equals to $\sum_{k=0}^{\lfloor \delta/(2K) \rfloor} d^{k \cdot K}$, which is equal to $(d^{\lfloor K \cdot \delta/(2K) \rfloor + 1} - 1)/(d^K - 1)$ and behaves like $d^{K \cdot \delta/(2K)}$. This signifies that $|C|$ augments exponentially with K. However because of the discretised parameter, $|C|$ is non-monotonically increasing with K.

The real nature of this approach is nevertheless not captured by the above formulas. The approximate method was developed in order to capture the locality of the interactions inside a MAS. And in fact the performance of this approach is directly tied to the nature of the MAS. With this approach a MAS where all agents share a common variable would each time perform a complete calculation over the cliques of the coordination. Therefore in such a case the approximate approach is not attractive compared to the exact one. However in a MAS were agents share only a few variables the graph G_R would likely be a stable. In this best case scenario, all computations are limited to the subgraphs connected to the local root to which the altered clique was attributed. Moreover such computations are done in a time proportional to $2K$ only, the performance bound remains however unchanged. The kind of MAS this approach aims at are therefore MAS whose interaction graph presents few centralised patterns.

Preliminary tests made over cliques' approximations indicate good overall behaviour, with approximate optimum very near the real optimum. However such good results are imputable to the structure of cliques' utility table. Those were filled according to a random uniform distribution. The bigger is a variable's cardinality size, the greater is the chance of finding a near-optimal value in any sub-clique of the clique. However worst case scenarios, with the approximated clique being forced to choose a local minima are still possible in this setting. This is why we introduced a secondary coordination phase in the approximate approach in order to lower the impact of such scenarios. It nevertheless remains that the applicability of this approach also depends on the very nature of agents' preferences.

The idea of using the multi-root approach recursively during the coordination of the roots naturally comes to mind. Using such an approach the total running time would be proportional to $(2K+1) \cdot (1 + \lfloor log(\delta_G)/log(K) \rfloor)$ which behaves like $K \cdot log(\delta_G)/log(K)$, where $\lfloor log(\delta_G)/log(K) \rfloor$ is the maximum number of recursive calls and $(2K+1)$ is the proportional time of each call. Such a running time is very interesting for a MAS procedure as it is scalable. It corresponds to a hierarchical decomposition of the global coordination problem in its decentralised components. However while it is interesting to take into account the natural decentralisation of a problem, a worst case behaviour happens when the problem is naturally centralised. In fact in such a setting all the cliques would all the same have to recompute their value. Adding superior levels of coordination has the consequence of slowing the computation, which is not as much a problem as it also exponentially decrease the quality approximations made at the upper levels, while the resulting performance over w can not be easily bounded.

5. CONCLUSION

We have presented in this paper the problem of efficient open MAS coordination using GAI-networks, and proposed two algorithms taking into account MAS decentralisation and a restricted notion of openness. This work constitutes a first step towards the study of the general case of openness, where not only the values of the cliques do change but also the structure of the junction tree. However, prior to such an extension good heuristics for selecting K and R so as to both minimise the total running time of the approximate algorithm and the number of cuts done over agents preferences have to be investigated, for it is the main concern regarding the performance bound. As the algorithm used for inference over GAI-networks is an instance of the sum-product algorithm, it would be therefore interesting to study how the approaches proposed in this paper could be extended to other kind of cooperative inference tasks instances of the factor graphs and the sum-product algorithm. Finally, concrete experimentation over real multi-agent systems is mandatory in order to evaluate the performance of those approaches.

6. REFERENCES

[1] P. Perny C. Gonzales. Gai networks for decision making under certainty. *IJCAI 05 – Workshop on Advances in Preference Handling*, 2005.

[2] R. Parr C. Guestrin, D. Koller. Multiagent planning with factored mdps. *Advances in Neural Information Processing Systems*, 2002.

[3] B. Milch D. Koller. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 2003.

[4] D. Koller D. Vickrey. Multi-agent algorithms for solving graphical games. *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[5] H.A. Loeliger F.R. Kschischang, B.J. Frey. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 2001.

[6] S. Singh M. Kearns, M. Littman. Graphical models for game theory. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2001.

[7] C.E. Guestrin M.A. Paskin. Robust probabilistic inference in distributed systems. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2004.

[8] S.L. Hakimi O. Kariv. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 1979.

[9] A. Petcu. A class of algorithms for distributed constraint optimization. 2007.

[10] A. Petcu and B. Faltings. S-dpop: Superstabilizing, fault-containing multiagent combinatorial optimization. *AAAI*, 05.

[11] A. Petcu and B. Faltings. O-dpop: An algorithm for open/distributed constraint optimization. *AAAI*, 2006.

[12] M. Yokoo P.J. Modi, W.S. Shen M. Tambe. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 2006.

[13] A. Tamir. An o(pn2) algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 1996.