Proceedings of the Fourth International workshop on

Optimisation in Multi-Agent Systems

3rd May 2011

(held in conjunction with AAMAS 2011)

Editors:

Alessandro Farinelli

Jesus Cerquides

Juan-Antonio Rodríguez-Aguilar

Sarvapali D. Ramchurn

Table of Contents

Note: The following paper has been intentionally excluded to avoid copyright issues: Ruben Stranders, Francesco Maria Delle Fave, Alex Rogers and Nicholas R. Jennings *U-GDL: A decentralised algorithm for DCOPs with uncertainty.*

Foreword

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 optimization 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 optimization 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, we initiated the workshop on Optimisation in Multi-Agent Systems (OPTMAS) in 2008 (and followed on every year till now) 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 optimization problems. In 2010 we collected all the best papers from previous editions of OPTMAS for a special issue of the Journal of Autonomous Agents and Multi-Agent systems due to appear in 2011.

This year's proceedings is composed of a number of contributions mainly focused on distributed optimisation and human-agent collaboration optimisation. As will be noted, the community is converging towards more realistic deployments of multi-agent systems, which is one of the main goals we set out to achieve initially.

Alessandro Farinelli

Jesus Cerquides

Juan-Antonio Rodriguez

Sarvapali D. Ramchurn

DGOPT: Dynamic Group Optimization to Find Better Group Formations in DCOPs

Elnaz Bigdeli, Maryam Rahmaninia, and Mohsen Afsharchi

Institute for Advanced Studies in Basic Sciences
Zanjan, Iran
{e_bigdeli,m_rahmani,afsharchim}@iasbs.ac.ir

Abstract. A substantial amount of study in multi-agent systems has focused on multi-agent coordination for over twenty years. Many challenges in multi-agent coordination can be modeled as Distributed Constraint Optimization (DCOP). Finding the optimal solution for a DCOP is NP-hard, so using incomplete algorithms that are faster are more desirable. Many incomplete algorithms decompose a DCOP to subgraphs to find solutions to it and maintain the partitioning of the DCOP unchanged during algorithm execution. These algorithms provide local optimal solutions. Decomposition of a DCOP has direct influence on the quality of solutions. With the popularity of incomplete algorithms, finding the best decomposition of a DCOP becomes a major issue. In this paper, we propose the first known learning algorithm by which the leader of each group optimizes its group with the purpose of increasing total utility. The leader agents learn to add/remove agents of their groups. This algorithm works dynamically to optimize the existing groups and we call it Dynamic Group Optimization algorithm (DGOPT). From quality, and convergence time point of view, DGOPT outperforms recent algorithms.

Keywords: Multi Agent Systems, Distributed Constraint Optimization, t-distance Optimality

1 Introduction

Multi-agent systems are a popular way to model complex interactions and coordination required to solve distributed problems. A multi-agent system is a network of agents used to perform distributed computation. Networks of agents are heterogeneous and not all agents have direct communication link to one another. Additionally, information is distributed throughout the network either due to privacy concerns or impractically of centralizing. In this network each agent is autonomous entity with local information and has ability to perform an action in cooperative situations in which agents collaborate to achieve a common goal.

Agents need to coordinate their activities to accomplish their collective goals. Distributed Constraint Optimization (DCOP) is a common formalism to represent multi-agent systems in which agents cooperate to optimize a global objective

[10, 13]. DCOP has been applied to different domains. DCOPs are able to model the task of scheduling meetings in large organizations, where privacy needs make centralized constraint optimization difficult [9]. DCOPs are also able to model the task of allocating sensor nodes to targets in sensor networks, where the limited communication and computation power of individual sensor nodes makes centralized constraint optimization difficult [11]. Finally, DCOPs are able to model the task of coordinating teams of unmanned vehicles in disaster response scenarios, where the need for rapid local responses makes centralized constraint optimization difficult [2].

There are two main categories for DCOP algorithms, complete and incomplete algorithms. Complete algorithms always find a configuration of variables that maximizes the global objective function. Adopt (Asynchronous Distributed OPTimization) [11] and DPOP (Dynamic Programming OPtimisation) [13] are two well known complete algorithms. There are lots of works which try to extend the ADOPT algorithm as a complete algorithm [4, 15]. The important point in complete algorithms is that finding DCOP solutions which maximize the global objective function is NP-hard. Some of recent works try to solve this problem [16, 17].

In contrast, incomplete algorithms find semi optimal solutions and do not guarantee to achieve global optimal solution. Algorithms such as Max-Sum [1], Distributed Arc Consistency [3] and KOPT [7] are in this category.

In the most of incomplete algorithms a network of agents is divided to groups in which a DCOP problem is solved locally [7, 10, 18]. The local attempt of agents in groups to solve DCOP leads to solving it globally, but the solution found is not the best.

KOPT and DALO algorithms are two examples of incomplete algorithms that divide the network of agents to subgroups to solve DCOP. k-optimal algorithms guarantee to provide solutions that cannot be improved by any group of k or fewer agents changing their decision. KOPT algorithm is the only incomplete algorithm which works for arbitrary k [7].

DALO is a novel asynchronous incomplete algorithm which works based on t-distance optimality [8, 18]. In DALO algorithm groups are formed based on the distance between nodes in the constraint graph instead of strict limits on group size. There are lots of incomplete algorithms to solve DCOP. The main concern in all of these algorithms is how to form groups because groups formation has direct influence on the quality of solution which is gained. We try to find better group formations through a dynamic approach. This approach works based on the contribution of each agent in the reward of the group.

The structure of the paper is as follows: In section 2, formal definitions of DCOP and t-distance optimality solutions are presented. Section 3, gives a general view of different group formations. In section 4, DALO algorithm and its main issues are described. The proposed algorithm is introduced in section 5. Detail description of dynamic group optimization is given in section 5.1. DGOPT algorithm is introduced in section 6. Experimental results of DGOPT algorithm

and its comparison with DALO algorithm are depicted in section 7. Finally, conclusion and future work are presented in section 8.

2 Background

In this section, we will provide some basic definitions about DCOP and t-distance optimality.

2.1 Distributed Constraint Optimization

A DCOP is defined by a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$, a set of discrete finite domains for each v_i; $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, and a set of constraints $\mathcal{C} = \{c_1, \ldots, c_q\}$. Each variable is controlled by a separate agent capable of communicating with other agents. A joint assignment $\mathcal{A} = \{a_1, \ldots, a_n\}$ specifies a value for each variable, in which a_i is the value of agent i. Each constraint includes a set of variables. A constraint defines a real-valued cost based on the values which each agent chooses for its variable. This paper holds in view binary constraints to avoid complexity; that is to say each constraint includes two variables. Thus, for each pair of variables (v_i, v_j), there is a cost function $\mathcal{F}_{ij} : \mathcal{D}_i \times \mathcal{D}_j \longrightarrow \Re$ which determines the value of a constraint. If there is no constraint between v_i and v_j, function $\mathcal{F}_{ij}$ will be 0. A cost function takes values of variables as an input and returns a value as a non-negative number for a given constraint. Utility of agent i for assignment $\mathcal{A}$ is:

$$\mathcal{U}_i(\mathcal{A}) = \sum_{v_j \in \mathcal{V}} \mathcal{F}_{ij}(a_i, a_j)$$
$$\text{Where } v_i \leftarrow a_i, v_j \leftarrow a_j, a_i, a_j \in \mathcal{A} \tag{1}$$

It means the utility of the i_{th} agent is the sum of the cost functions of all the constraints to which an agent belongs.

The goal is to choose values for variables such that a given objective function is maximized. The objective function is described as the sum over a set of cost functions, or valued constraints. As a result, the objective is to maximize:

$$\mathcal{R}(\mathcal{A}) = \sum_{(v_i, v_j) \in \mathcal{V}} \mathcal{F}_{ij}(a_i, a_j)$$
$$\text{Where } v_i \leftarrow a_i, v_j \leftarrow a_j, a_i, a_j \in \mathcal{A} \tag{2}$$

$\mathcal{R}(\mathcal{A})$ is a solution quality for an assignment $\mathcal{A}$ [11, 12].

Figure 1 shows an example of DCOP with 6 variables and 7 constraints with the same cost function. The optimal assignment for this DCOP is $\mathcal{A} = (1, 1, 1, 1, 1, 1)$.

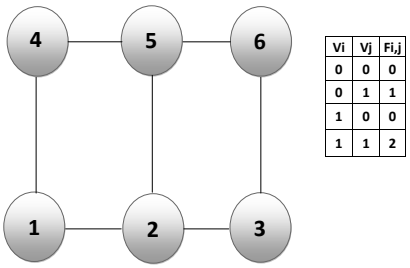


Fig. 1. An example DCOP with six binary variables. Each constraint has the same cost function.

2.2 *t*-distance Optimality

Definition 1 *For two different assignments $\mathcal{A}$ and $\mathcal{A}'$:*

$$\mathcal{D}(\mathcal{A}, \mathcal{A}') = \{v_i \in \mathcal{V} \parallel a_i \neq a_i' \ , \ v_i \leftarrow a_i \in \mathcal{A}, v_i \leftarrow a_i' \in \mathcal{A}'\} \tag{3}$$

Put simply, $\mathcal{D}$ is a deviating group between two assignments $\mathcal{A}$ and $\mathcal{A}'$.

Definition 2 *For a pair of variables v_i and v_j, let $\mathcal{T}(v_i, v_j)$ be the shortest distance between them in the constraint graph. Let $\Phi_t(v_i) = \{v_j \| \mathcal{T}(v_i, v_j) \leq t, v_i, v_j \in \mathcal{V}\}$ denote a set of variables that can be reached from v_i within t hops.*

Definition 3 *A DCOP assignment $\mathcal{A}$ is t-distance optimal if $\mathcal{R}(\mathcal{A}) \geq \mathcal{R}(\mathcal{A}')$ for all $\mathcal{A}'$, where $\mathcal{D}(\mathcal{A}, \mathcal{A}') \subseteq \Phi_t(v_i)$ for some $v_i \in \mathcal{V}$ [8, 18].*

Example: Consider the graph in Figure 2. Given $t = 1$, 1-distance groups for all variables will be: $\Phi_1(v_1) = \{v_1, v_2, v_3\}$, $\Phi_1(v_2) = \{v_1, v_2, v_4, v_5\}$, $\Phi_1(v_3) = \{v_1, v_3, v_4\}$, $\Phi_1(v_4) = \{v_2, v_3, v_4, v_5\}$, $\Phi_1(v_5) = \{v_2, v_4, v_5\}$.

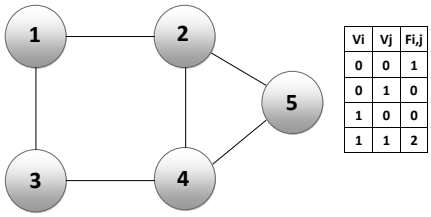


Fig. 2. An example DCOP with five binary variables. Each constraint has the same cost function.

3 Structures of Groups

In a DCOP, the problem is solved by dividing DCOP into groups, and then, all the agents in a group cooperate to maximize the objective function. In other words, they create a coalition in groups to maximize the objective function. A group structure is a partition of the overall set of agents into sub groups. DCOP division in section 2.2 with $t = 1$ for the graph in Figure 2, is a possible structure for this network of agents. The structure of groups by this division is:

$$GS = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4, v_5\}, \{v_1, v_3, v_4\}, \\ \{v_2, v_3, v_4, v_5\}, \{v_2, v_4, v_5\}\}$$

Obviously, by using different t for each group, other structures are gained.

Given a network of agents NET $= (\mathcal{A}g, \mathcal{F})$ with a set of agents $\mathcal{A}g$ and a set of cost functions $\mathcal{F}$, the optimal group structure GS^* is given in the following formula:

$$GS^* = \underset{GS \in all\ possible\ group\ stuctures}{\arg\max} U(GS) \tag{4}$$

Where

$$U(GS) = R_{GS}(A^*) \tag{5}$$

It indicates $U(GS)$ is the reward value for assignment $\mathcal{A}^*$ which is the best assignment reached by applying an incomplete algorithm for a DCOP.

Finding the best structure is impossible because the number of structures that can be created in a graph is exponential.

4 DALO Algorithm and Issues

DALO algorithm, as an incomplete algorithm, was introduced by Yin [18]. It is an asynchronous algorithm for DCOP based on t-distance optimality.

DALO algorithm has three phases. In phase one, each agent sends a message containing all its constraints to agents in a distance of t hops. Then, it broadcasts its initial value to a distance of $t + 1$ hops in a separate message. In phase two, based on the information gathered in the previous phase, all the leaders compute a new optimal assignment using a centralized variable elimination algorithm in parallel. In phase three, if the new assignment improves the utility of a group, the group leader attempts to set the new assignment. There might be conflicts among overlapping groups while all leaders try to set their assignments. The conflicts are resolved by an asynchronous locking and commitment protocol.

4.1 DALO Issues

Although DALO is an effective algorithm to solve DCOP problems, it suffers from some drawbacks. 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 [18]. Using distance as a criterion to create groups may produce groups with large number of nodes; especially, when there are hub nodes with many connections or subgraphs which are densely connected.

As it is explained in DALO algorithm in phase two, a complete algorithm is used to solve DCOP. All group leaders compute new optimal assignments for their groups in parallel. A leader node uses a centralized variable elimination algorithm to find the best assignment for the local group. Variable elimination algorithms are complete algorithms with exponential computational complexity in the number of agents. By increasing t, the number of agents in a group will increase and using a complete DCOP solver will not be tolerable from size and space point of view. To solve this problem, instead of using a centralized variable elimination algorithm, we use a genetic approach in phase two which is discussed in [14].

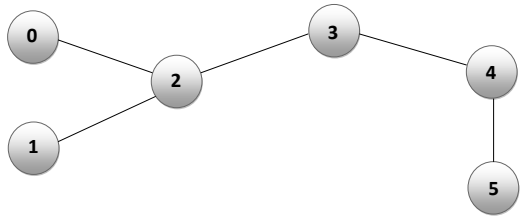
One of the significant problems in DALO algorithm and some other incomplete algorithms is how to form groups to reach the highest utility. Groups formation has direct influence on the quality of solutions for a given DCOP. Using some formations, the algorithm cannot improve the quality through increasing the number of rounds [7]. On the other hand, by changing the group formation new values may be set and the quality may improve. This problem stems from the conflicts among groups. The presence of some agents in some groups does not let a group improve its local solution, since these agents are common agents among different groups and some of them do not commit to the assignment of many groups to which they belong.

Changing groups leads to solution variation. Finding the best group formation, which the best solution could gained from, is very difficult and in some cases is impossible. With the purpose of finding the best group formation all possible formation should be considered and after comparing the results the best one is chosen. As it is clear, it is impossible in networks with large number of agents.

This paper introduces a distributed approach to improve groups formation. This method works based on the contribution of each agent in the reward of the group. To shed light on the problem an example is given in the next section.

Example Consider the graph in Figure 3. The cost function for each constraint is given. Groups are formed using $t = 1$. Active agents are shown in bold and passive agents are shown in italic. Here after we use the terms active and passive agent more. Hence, it is worthwhile to define them here. Active agents are those that can change their value to the value which leaders send to them. In contrast, passive agent are those agents in the boundary of group whose values do not change by the leader of group and their values remain constant during algorithm execution.

$\mathcal{G}_0 = \{\mathbf{0}, \mathbf{2}, \mathit{1}, \mathit{3}\}, \mathcal{G}_1 = \{\mathbf{1}, \mathbf{2}, \mathit{0}, \mathit{3}\}, \mathcal{G}_2 = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathit{4}\}, \mathcal{G}_3 = \{\mathbf{2}, \mathbf{3}, \mathbf{4}, \mathit{0}, \mathit{1}, \mathit{5}\}, \mathcal{G}_4 = \{\mathbf{3}, \mathbf{4}, \mathbf{5}, \mathit{2}\}, \mathcal{G}_5 = \{\mathbf{4}, \mathbf{5}, \mathit{3}\}$



0	2	F
0	0	10
0	1	0
1	0	0
1	1	0

2	3	F
0	0	2
0	1	0
1	0	2
1	1	0

1	2	F
0	0	0
0	1	8
1	0	0
1	1	0

4	5	F
0	0	8
0	1	0
1	0	0
1	1	0

3	4	F
0	0	0
0	1	0
1	0	8
1	1	0

Fig. 3. DCOP examples

All leaders $\mathcal{L} = \{0, 1, 2, 3, 4, 5\}$ compute the best assignment for their group members by starting from initial value 1 for all agents. Among these leaders, leader 4 can set its assignment. Since, leader 4 sets its assignment, the other leaders cannot set their assignments. Therefore, DCOP assignment will be $A = (1, 1, 1, 1, 0, 0)$. The utility of DCOP will be $U(A) = 16$.

Agent 3 is the common agent in groups $\mathcal{G}_2$, $\mathcal{G}_4$ and also is the active agent in these two groups. In computing the best assignment the value given to agent 3 by leader 2 is 0 but the value given by leader 4 to the same agent is 1. Based on the rule in DALO algorithm, an agent is committed to the group which has the highest utility. Hence, based on cost functions in Figure 3, agent 3 chooses the value given by leader 4.

Among groups $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2$, without considering other groups of graph, group $\mathcal{G}_2$ can set its assignment, but other groups cannot because all the common agents in these groups commit to the values given by leader 2. On the other side, due to the presence of agent 3 which commits to the value of leader 4, leader 2 cannot set its assignment. Consequently, the presence of agent 3 makes groups assignment stay unchanged. It is needless to say quality does not enhance as result of this presence.

According to the description above among all groups, group $\mathcal{G}_4$ changes its assignment and all other groups stay in their initial values. Consider agent 3 is removed from group $\mathcal{G}_2$. This change having been incorporated, group $\mathcal{G}_2$ and group $\mathcal{G}_4$ set their assignments simultaneously which leads to utility enhancement.

This example clarifies the main problem in algorithms that use a fix group formation in which the utility of the solution does not increases without changing group formation. Worded differently, it shows that the presence of some

active/passive agents creates problems in setting new assignments and adding or removing them to or from some groups solves the problem.

For the above reasons, we focus on the impact of each agent in each group. We use a novel algorithm which tries to estimate the impact of agents in groups. This algorithm is an efficient and distributed method to change groups dynamically. The algorithm considers the impact of active agents in a group as well as passive agents. Moreover, each group leader keeps only the local group information to run the algorithm which makes the communication bandwidth and storage requirement low.

5 Embedding Group Optimization in Solution Procedure

The solution procedure starts from a random initial assignment and monotonically improves the solution quality. To have our discussion simple we divide the procedure into four phases: initialization, groups optimization, computing the best assignment and implementing assignments.

- **Initialization:** At first, every agent sends a message containing all its constraints to all agents in distance of t hops from it. Then, it chooses an initial value from its domain and broadcasts it to agents in distance of $t + 1$ hops. In this phase, a leader starts to construct its group. Given t, all agents whose distance of center node are lower than t will be a member of the group. The additional hop in sending a message is for boundary nodes. The nodes in the boundary of a group are considered static in computing the best assignment.
- **Groups optimization:** In phase two, some leaders are selected and optimize their groups to facilitate the achievement of a better solution for a DCOP problem. Detailed description of optimization algorithm is given in section 5.1.
- **Computing the best assignment:** In this phase, all the leaders compute a new optimal assignment using a centralized variable elimination algorithm in parallel. A leader agent finds a new value for active agents in a group considering the fact that passive agents stay unchanged.
- **Implementing assignments:** Each agent belongs to different groups and receives various assignments from different leaders, and lastly every agent commits to a group with the highest utility. To resolve the conflict among overlapping groups, a method is used for resolving conflict described in DALO algorithm [18].

Phase one is done just once. Since the computation complexity of phase two is high, this phase is executed after each m rounds. The value of parameter m depends on the size of graph and is specified through experiment. Optimizing group is our main contribution in this paper and so we provide our deep discussion about this issue in the next subsection. Phase three and four are executed in each round in all groups in parallel. The algorithm stops running if it converges to a value.

5.1 Optimizing Groups

As it is mentioned before, we try to find a group formation by which solutions with higher quality are gained. To this end, a leader tries to change its group by adding /removing some agents to/from its group. When a leader decides to add a passive agent to the group, it changes it to an active one. In contrast, to remove an agent a leader makes an active agent passive.

How to add or remove agents is the major problem. A criterion should be introduced to use in adding or removing the agents. We utilize a marginal contribution concept to change groups.

Definition 4 *Let $\mu_i(\mathcal{G})$ be the marginal contribution of $\mathcal{A}gent_i$ to the group $\mathcal{G}$ which is computed by adding or removing it from $\mathcal{G}$.*

$$\mu_i(\mathcal{G}) = R(A') - R(A) \tag{6}$$

A' is the best assignment of group $\mathcal{G} \subseteq$ all agent $\cup\, \mathcal{A}gent_i$ and A is the best assignment of group $\mathcal{G} \subseteq$ all agent$\backslash\mathcal{A}gent_i$.

We use this concept in group formation. In each group of a DCOP, utility of a group before and after removing (adding) an agent is computed and if the absence (presence) of an agent increases the local utility, we try to change the group by removing (adding) an agent.

There are three main issues in groups optimization. The first one is that the decision about any changes cannot be made through group information and decision about the change should be made using the information of the whole DCOP, but we do not have the global information of the graph in each group. The second issue is about the method of group alternation as adding and removing an agent causes some other agents to join or leave the group. Changing all groups is not efficient which forces us to choose some leaders to change their groups. This is the source for the third issue. All these problems and our proposed solutions are discussed in the following sections in more detail. From now on, we call the group which we try to change *the target group*.

Local View of the Leader Agent Changing a group has an effect on the whole DCOP, because by removing (adding) agents the assignment for the members changes. As a matter of fact, the new assignment of this group has direct influence on other groups. The new values may or may not enable some other groups set their assignments. For that reason, a leader agent should be aware of the status of other groups and whether or not they set their new assignment through the new change. A group leader can firmly claim that the change has positive effect if it has global information about the graph. It is obvious that the leader does not have such information.

Based on the description above, due to the connections among groups and propagation of the change in the whole DCOP, to get the best decision, a leader agent should be informed of their adjacent leaders and decide what happens in

the whole DCOP after the new change. But as we know, it is impossible to solve a problem distributively.

We solve the problem by creating a local view for each leader. The local view of the leader of target group is a subgraph consists of the target group and all its adjacent groups. In the process of decision making about the new change in the target group, the leader uses local view. There is no need to know all the members of the adjacent groups. Because the leader of the target group should only be in contact with the leaders of the adjacent groups. Finding the leader of any of adjacent groups is very easy because these leaders are the active agents of the target group.

When a leader changes its group, it computes a new assignment for the new group and sends the new values to all group members. All agents in this group receive new assignment and decide about commitment to the group again. Then, all the groups in the local view of the leader of the target group use the new assignment and decide about implementation of their assignments. Adjacent groups do not compute new assignments and they just receive a message from agents which commit to other groups. The leader of the target group sends a message to its adjacent leaders to be aware of the utility of the adjacent groups. All adjacent leaders send back their utilities to the central leader.

Add/Remove Agents of a Group A leader has information about its members including active and passive agents. Because of the limited information of the leader node, the agent we try to add to the group should be chosen from passive agents of the target group. All agents with direct link to the added agent and not belonging to the group will be considered as passive agents of the new group.

In removing an agent from a group, we choose an active agent and make it a passive one, but all passive agents which are connected to the removed agent should be removed from the group as well. We cannot remove all of these agents because there are some agents among these agents which are connected to the group via other active agents. Therefore, in removing an agent from a group, agents connected to a group just by the removed agent will be removed and all other agents will remain by means of other active agents in the group.

Choosing Groups to be Optimized One of the main issues in this new approach is to choose groups to be optimized. We can apply the new approach to all groups, but it has some problems. The first is that using the new approach in solving a DCOP, the computational cost increases and consequently optimizing all groups is not tolerable from computational point of view. The second problem is related to coordinating the decision of all leaders in changing their groups. Since leaders do not have global information of a DCOP, their decision about the changes in their groups may have conflicts and optimizing all groups will be useless.

In line with aforementioned description, some leaders should be chosen to optimize their groups. Choosing groups optimizing of which provides us with the

best result is impossible. We use a simple and cost effective method to select groups. This approach is derived from a partial approach introduced in [6]. Using this approach all leaders should try to change their groups, at least once, during algorithm execution. As a matter of fact, the changes in a group formation influences the whole DCOP and increases the computational cost. Hence, optimizing groups is done after each m rounds.

Consider there are n agents in a network. Consequently, there are n groups in a network. We define the index set $\mathcal{L} = \{1, 2, \dots, n\}$. The index set $\mathcal{L}$ is divided into h subsets $\mathcal{S} = \{S_1, \dots, S_h\}$. Each subset contains leaders ID which should optimize its groups. After each m rounds, a subset, S_i, is selected and leaders in this subset try to optimize their groups. The main problem is to assign leaders to subsets. Finding the best division is not computable within limited time.

We use a simple approach called sequential approach. In sequential approach, in the first round, S_1 is selected, in round m subset S_2, and in round mh subset S_h is selected; in round $m(h+1)$, subset S_1 is selected again. So, after mh rounds, all subsets will be selected only once.

To specify groups to be optimized, we use a simple rule. In round r, a leader with ID ℓ checks if:

$$\ell \ \% \ h = r \ \% \ h \ , \ where \ r \ \% \ m = 0 \tag{7}$$

Then, this leader tries to optimize its group. The number of groups which should be optimized and the rounds in which we optimize groups are found through experiment. There is no specific rule to do so.

6 DGOPT Algorithm

In this section, we explain DGOPT in more detail. As it is obvious DGOPT adds a new step to DALO. For the sake of simplicity we just emphasize on the new step when we refer to DGOPT. We divide the algorithm into 3 steps to have more concentration on our explanation. This algorithm is applied to groups which are selected based on descriptions in the previous section. The following algorithm represents the process of removing an agent. The process of adding is much the same way.

– **Local information gathering:** At first, the leader node finds its adjacent leaders which are the active agents of its group. The leader of target group stores the utility of its adjacent groups which are computed before optimizing its group. Since the leaders in the local view are the active agents of the target groups, obtaining information about their groups is not very time-consuming. Next few lines yield the justification of why this process is not time-consuming. To commit to the new assignment, some messages pass among leader and its active agents. We can include the utility of adjacent groups in the messages by which active agents inform the leader whether or not they have committed to the new assignment. Therefore, there is no need

to send and receive more messages and we can include the information for decision making in messages which are exchanged among leader and active agents to set the new assignment. Based on this information, we sum up the utility of the target group and the groups in its local view in line 6 to have the utility of groups in local view before changing the target group.

Algorithm 1

(∗ Remove an Agent From a Group ∗)

1. Target group $\mathcal{G}$;
2. Create Local view();
3. Local Information Gathering();
4. $R_1 = 0$;
5. **for** $i \leftarrow 1$ **to** Number of Groups in Local View
6. **do** R_1=R_1+utility(group(i));
7. **for** $i \leftarrow 1$ **to** some randomly chosen active agents
8. **do** $R_2 = 0$;
9. Temporary Remove Agent(agent(i));
10. Compute Utility($\mathcal{G}$);
11. **for** $j \leftarrow 1$ **to** Number of Groups in Local View
12. **do** R_2=R_2+utility(group(j));
13. **if** $(R_2 - R_1) > 0$
14. **then** Remove Agent Permanently();
15. **return** $\mathcal{G}$;

- **Changing group temporarily and computing the new assignment:** The leader removes an active agent temporarily. The leader agent makes the active agent a passive one and removes all agents connected to the group by this agent. If the agent is connected to the group by other active agents, we do not remove it. After removing an agent, the leader node re-computes the best assignment in the new group. All leaders of adjacent groups just decide about the implementation of their assignments by the new change and they send the utility of their groups to the leader of the target group. We emphasize that by the above justification the adjacent leaders are not in need of sending new messages to inform the leader of the target group of their utilities. The leader agent sums up its new utility and its adjacent leaders' utilities again in line 12.
- **Computing marginal contribution:** In this step, marginal contribution of the removed agent is computed. The positive marginal contribution of the removed agent shows that the new formation increases the utility, but if the marginal contribution is negative, the change is not promising. If the change is promising the group is changed permanently in line 14.

The first phase is executed just once. The other phases are repeated for all selected active agents.

7 Experimental Results

In this section we put forward some experimentations to show the efficiency of our DGOPT algorithm. We setup our experiments on some graphs with different densities and the same size. Before presenting our results we introduce our three evaluation metrics which the algorithms are compared based on them, namely, quality of solution, number of rounds and (Gain,#Locked Variables).

- **Quality of Solution:** The primary aim of this paper is to construct groups to reach the solution with higher quality. The quality of solution is computed according to equation 2. Based on this definition, a solution with higher reward is more qualified [5, 7, 12].
- **Number of Rounds:** A round is one unit of algorithm progress in which all agents perform any required computation. After some rounds, the solution reached by algorithm does not change. In this case, the algorithm converges to the best possible solution. In the evaluation, we consider the number of rounds required to converge. This metric is a convenient, standardized metric for estimating the performance of a DCOP algorithm [5].
- **(Gain,#Locked Variables):** Versus to above mentioned metrics which are used to evaluate the performance of DCOP algorithms in the whole DCOP, tuple $(Gain, \#Locked\ Variables)$ analyzes the performance of the algorithm on local groups. The gain is the quality of group and the $\#Locked\ Variables$ is the number of variables that are locked to set the new assignment.

7.1 Results

The result that we are reporting is based on some random graphs with four different densities $\mathcal{D} = \{0.2, 0.4, , 0.6, 0.8\}$. All graphs used in our experiment have the size 50 with different densities and structures. Variables have a binary domain and rewards are integers drawn from $[1, 500]$. In the experiment we generated 20 random graphs with different structures while kept the size and density the same. The solution quality shown in the following figures are the average quality that gained from these graphs.

Both algorithms start from a same random initial assignment. The stopping criterion is also defined in a same for both algorithms. The algorithm stops running whenever all groups do not tend to change their assignments because there is no new assignment to increase the utility of groups.

We set parameter t to 2, h to 3, and m to 5 respectively. Determination of the exact values of h and m is made just by experiment and we set the parameters to the values which have the more desirable results. More discussion related to determination of h and m can be found in [6].

In our first experiment we compare the solution quality of our DGOPT algorithm and DALO. Obviously, algorithm that achieves a final solution of higher quality in a lower number of rounds is more desirable. Figures 4 through 7 show that the solution quality increases by DGOPT algorithm in comparison with DALO. For instance in Figure 4 the final solution quality for graphs with

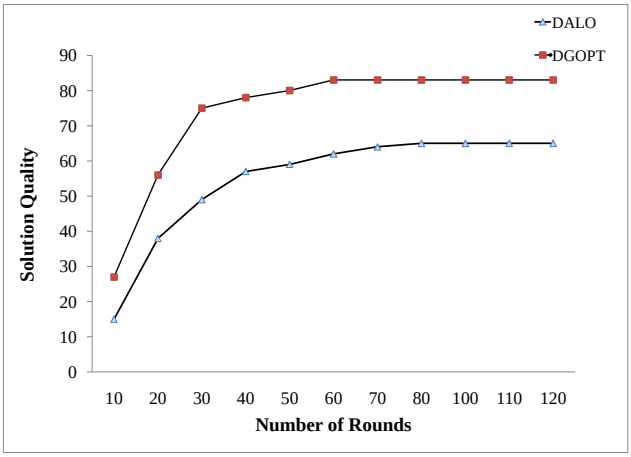


Fig. 4. Solution quality: DGOPT vs DALO for graphs with density 0.2

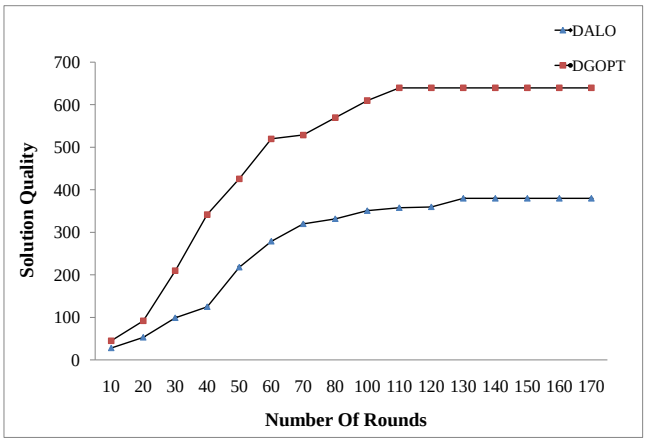


Fig. 5. Solution quality: DGOPT vs DALO for graphs with density 0.4

$\mathcal{D} = 0.2$ using DALO is 65, but using DGOPT the quality in the same graphs is 82. Moreover, after group alteration through DGOPT, there will be a boost in the solution quality; these increases end in the algorithm convergence to a higher solution quality in lower number of rounds in comparison with DALO. As an example, consider the diagrams in Figure 6, DGOPT converges after 150 rounds and DALO converges after 195 rounds.

The results also show that the DGOPT algorithm is even more efficient on dense graphs. For example, the maximum difference in solution quality for graphs with $\mathcal{D} = 0.2$ is almost 30, but the maximum difference for graphs with $\mathcal{D} = 0.4$

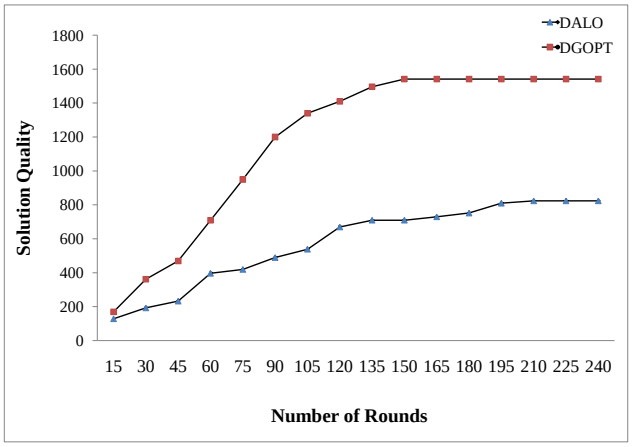


Fig. 6. Solution quality: DGOPT vs DALO for graphs with density 0.6

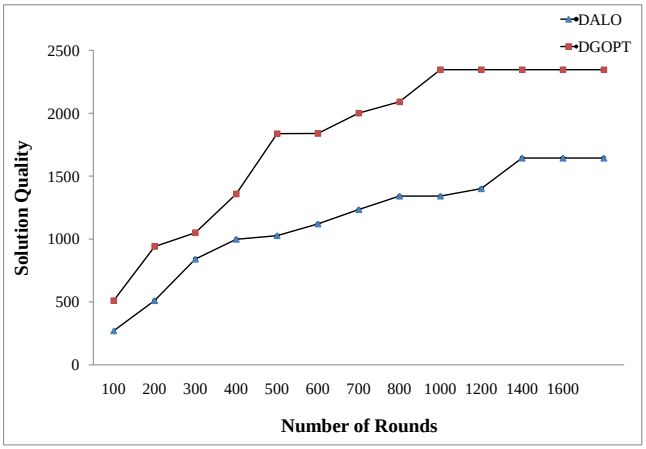


Fig. 7. Solution quality: DGOPT vs DALO for graphs with density 0.8

is almost 300. It is clear that the groups in dense graphs have more number of agents in comparison with sparse graphs. Accordingly the overlap among groups increases and there will be more number of agents which are common among groups. In this case, there will be more number of agents which do not allow a leader to set its assignment by committing to other groups. Overally, the results in our experiment show that the quality of solution increases 43% and the number of round decreases 21%, on average. It can be concluded that using DGOPT algorithm, solutions with higher quality are gained in a lower number of rounds.

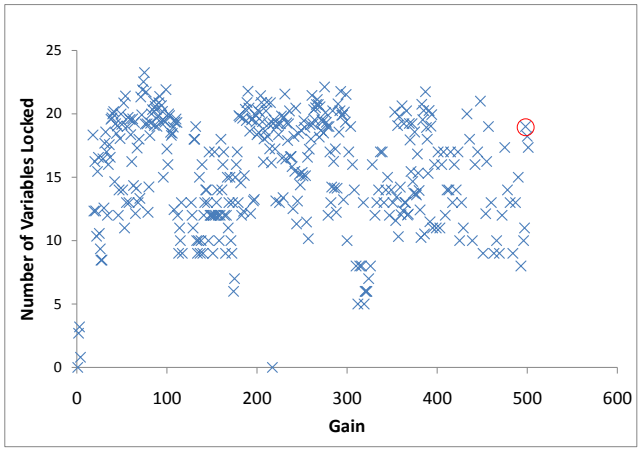


Fig. 8. (Gain,#Locked Variables) for DALO

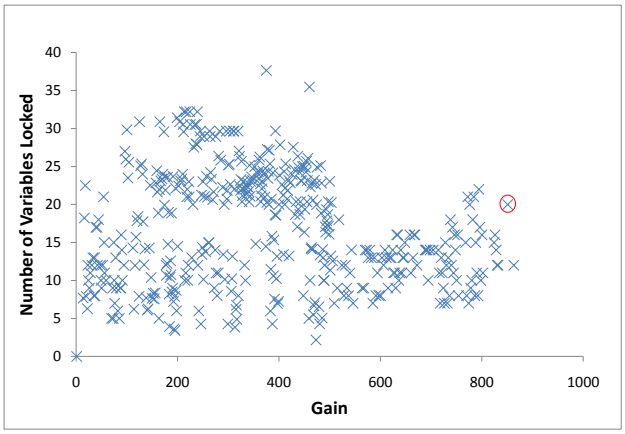


Fig. 9. (Gain,#Locked Variables) DGOPT

To further understand and compare the performance of DGOPT and DALO, we provide an analysis on local group changes. In each group, the leader locks some of the variables and if all group members commit to the new assignment, it will be set. By setting the new assignment, the utility of group, which we call it gain, will change. The $(Gain, \#Locked\ Variables)$ pair is used as a metric to compare DCOP algorithms in [18]. It is a proper metric to compare the effect of different group formations in solving DCOPs. The more the number of locks,

the more the number of conflicts. Hence, groups with lower number of locks and larger gain are more preferred.

To compare the algorithms, we depict the result for graphs with size 50 and density 0.4. As it is declared in Figure 8 DALO never achieves a gain larger than 500 and barely locks more than 20 variables. On the other hand, DGOPT achieves gain 800 by locking more number of variables. For example as it is specified in the Figures 8, 9 by locking 20 variables DALO achieves gain 500 while DGOPT can achieve gain 850.

In Figure 8, the congestion is on the value 18 which indicates that most of groups locked 18 variables. On the contrary, as it is depicted in Figure 9, the congestion is on the value 23. The difference in the number of locked variables is not very much, but the quality improvement is considerable. Hence, by slight increase in the number of variables better solutions are gained. Our experimentations show hat DGOPT outperforms DALO both in term of solution quality and the number of rounds that this quality is achieved.

8 Conclusion

As it is explained in this paper some group formations are not very efficient to solve DCOP problems and lead to solution with lower quality. In this paper, we proposed a distributed dynamic algorithm to optimize groups in a DCOP. The purpose of this algorithm is to find better group formations to reach higher solution quality. This algorithm achieves solutions with higher quality in low number of rounds. Moreover, by slight increase in the number of locked variables in groups, solution quality increases considerably. The proposed algorithm can be applied to other incomplete DCOPs by slight modification. We are planning to extend the algorithm by using agents previous interactions to improve groups.

References

1. Aji and R. McEliece. The generalized distributive law. ieee transactions on information theory. *IEEE Transactions on Information Theory*, pages 325–343, 2000.
2. A. Chapman, R. A. Micillo, R. Kota, and N. Jennings. Decentralised dynamic task allocation: A practical game-theoretic approach. *the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, pages 915–922, May 2009.
3. M. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *the Proceedings of the 20th Internation Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 68–73, 2007.
4. J. Davin and P. Modi. Hierarchical variable ordering for multiagent agreement problems. In *the Proceedings of AAMAS*, pages 1433–1435, 2006.
5. J. Davin and P. J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *the Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1057–1063, July 2005.

6. E.Bigdeli, M.Rahmaninia, and M.Afsharchi. Pkopt: Faster k-optimal solution for dcop by improving group selection strategy. In *the proceeding 22th international conference on tools with Artificial Intelligence (ICTAI)*. October, July 2010.
7. H. Katagishi and J. P. Pearce. Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility. In *CP Workshop on Distributed Constraint Reasoning*, September 2007.
8. C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *the Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, May 2010.
9. R. Maheswaran, E. Bowring, J. Pearce, P. Varakantham, and M.Tambe. Taking dcop to the real world: efficient complete solutions for distributed multi-event scheduling. In *the Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, pages 310–317, 2004.
10. R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society, 2004.
11. P. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt:asynchronous distributed constraint optimization with quality guarantees. *ARTIFICIAL INTELLIGENCE*, 16(1-2):149–180, 2005.
12. J. P. Pearce, M. Tambe, and R. T. Maheswaran. Solving multiagent networks using distributed constraint optimization. *AI Magazine*, 28(3):47–66, September 2008.
13. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *the Proceedings of the International Joint Conference on Artificial Intelligence*, pages 266–271, aug 2005.
14. M. Rahmaninia, E. Bigdeli, and M. Afsharchi. Solving dstributed constraint optmazation problem: An evalutionary approach. In *the Proceedings of International Conference on Agents and Artificial Intelligence (ICAART)*, January 2011.
15. M. Silaghi and M. Yokoo. Dynamic dfs tree in adopt-ing. In *the Proceedings of AAAI*, pages 763–769, 2007.
16. W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. In *the Proceedings of AAMAS*, pages 591–598, 2008.
17. W. Yeoh, X. Sun, and S. Koenig. Trading off solution quality for faster computation in dcop search algorithms. In *the Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 354–360, 2009.
18. Z. Yin, C. Kiekintveld, A. Kumar, and M. Tambe. Local optimal solutions for dcop: New criteria, bound, and algorithm. In *AAMAS 2009 Workshop on Optimization in Multi-Agent Systems*, May 2009.

Balanced Exploitation and Exploration for Max-sum Distributed Constraint Optimization

Hilla Peled and Roie Zivan,
Industrial Engineering and Management department,
Ben Gurion University of the Negev,
Beer-Sheva, Israel
{hillapel,zivanr}@bgu.ac.il

Abstract. Distributed Constraint Optimization Problems (DCOPs) are NP-hard and therefore most recent studies consider incomplete (local) search algorithms for solving them. Specifically, the Max-sum algorithm has drawn attention in recent years and has been applied to a number of realistic applications. Unfortunately, in many cases Max-sum does not converge. When problems include cycles of various sizes in the factor graph upon which Max-sum performs, the algorithm does not converge and the states that it visits are of low quality.
In this paper we advance the research on incomplete search for DCOPs by: (1) Proposing a version of the Max-sum algorithm that operates on an alternating directed acyclic graph (Max-sum_AD), which guarantees convergence. (2) Proposing exploration methods that allow the algorithm to escape the high quality state to which it converges. Our empirical study reveals the improvement in performance of the proposed exploitive algorithm when combined with exploration methods, compared with the performance of the standard Max-sum algorithm.

1 Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general model for distributed problem solving that has a wide range of applications in Multi-Agent Systems and has generated significant interest from researchers [7, 8, 13, 10].

A number of studies on DCOPs presented complete algorithms [7, 8, 4]. However, since DCOPs are NP-hard, there is a growing interest in the last few years in local (incomplete) DCOP algorithms [6, 13, 14, 11, 12]. Although local search does not guarantee that the obtained solution is optimal, it is applicable for large problems and compatible with real time applications.

The general design of the state of the art local search algorithms for DCOPs is synchronous. In each step of the algorithm an agent sends her assignment to all her neighbors in the constraint network and receives the assignment of all her neighbors. They differ in the method agents use to decide whether to replace their current value assignments to their variables, e.g., in the max gain messages algorithm (MGM) [6], the agent that can improve her state the most in her neighborhood replaces her assignment. A stochastic decision whether to replace an assignment is made by agents in the distributed stochastic algorithm (DSA) [13].

An incomplete algorithm that does not follow the standard structure of distributed local search algorithms and has drawn much attention recently is the Max-sum algorithm [3]. In contrast to standard local search algorithms, agents in Max-sum do not propagate assignments but rather calculated utilities (or costs) for each possible value assignment to their neighboring agents' variables. The general structure of the algorithm is exploitive, i.e., the agents attempt to compute the best costs/utilities for possible value assignments according to their own problem data and recent information they received via messages from their neighbors.

The growing interest in the Max-sum algorithm in recent years included its use for solving DCOPs representing various multi-agent applications, e.g., sensor systems [11] and task allocation for rescue teams in disaster areas [9]. In addition, a method for approximating the distance of the solution found by Max-sum from the optimal solution for a given problem was proposed [2]. This version required the elimination of some of the problem's constraints in order to reduce the DCOP to a tree structured problem which can be solved in polynomial time. Then, the sum of the worst costs for all eliminated constraints serves as the bound on the approximation of the optimal solution.

Previous studies have revealed that Max-sum does not always converge to a solution [3]. In fact, in some of the cases where it does not converge, it traverses states with low quality solutions and thus, at the end of the run the solution reported is of poor quality. This pathology occurs when the constraint graph of the problem includes cycles of various sizes. Unfortunately, many DCOPs which were investigated in previous studies are dense and indeed include such cycles (e.g., [7, 4]). Our experimental study revealed that for random problems, for a variety of density parameters from as low as 10%, Max-sum does not converge.

An attempt to cope with the in-convergence of Max-sum was proposed in [3]. It included the union of groups of agents to clusters of adjacent agents represented by a single agent in the cluster. The constraints between the agents in the cluster were aggregated and held by the agent representing the cluster. Thus, it required that some constraints would be revealed in a preprocessing phase to agents which are not included in the constraints (the constraint between agents A_1 and A_2 is revealed to agent A_3). The amount of information that is aggregated is not limited and in dense problems can result in a single agent holding a large part of the problem's constraints (partial centralization). In this work we avoid such an aggregation of the problem's data in a pre-processing phase and propose algorithms and methods that solve the original DCOP (as the standard Max-sum algorithm does).

In this paper we contribute to the understanding of incomplete search for DCOPs by:

1. Proposing a new version of the Max-sum algorithm which uses an alternating directed acyclic graph (DAG). The proposed algorithm (Max-sum_AD) avoids cycles by performing iterations of the algorithm in which messages are sent according to a predefined order. In order not to ignore constraints of the DCOP, after a number of iterations which guarantees the convergence of the algorithm, the order from which the direction of the DAG is derived is reversed. Then, the algorithm is performed on the reversed DAG until it converges again. We prove that the maximal number of iterations in a single direction required for the algorithm to converge is equal

to the longest path in the DAG, l (linear in the worst case). Thus, by performing l iterations in each direction we converge to a solution after considering all the constraints in the DCOP.

2. Proposing exploration heuristic methods for Max-sum_AD. The proposed methods allow the algorithm to converge to different solutions of high quality. By using the algorithm within the anytime framework, proposed for local search on DCOPs in [14], we can select the best among these solutions to be reported by the algorithm at the end of its run. To best of our knowledge, no exploration methods were proposed for Max-sum to date. Thus, we are the first to balance between exploration and exploitation of the Max-sum algorithm. Our empirical study demonstrates the success of this balanced performance in comparison with the standard Max-sum algorithm.

The rest of this paper is organized as follows: DCOPs are presented in Section 2. Section 3 presents the standard Max-sum algorithm. The Max-sum_AD algorithm is presented in Section 4. Section 5 presents exploration methods for the Max-sum_AD algorithm. Section 6 includes an evaluation of the proposed algorithm and exploration methods. Our conclusions are presented in Section 7.

2 Distributed Constraint Optimization

A $DCOP$ is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, ..., A_n$. $\mathcal{X}$ is a finite set of variables X_1,X_2,...,X_m. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains D_1, D_2,...,D_m. Each domain D_i contains the finite set of values which can be assigned to variable X_i. We denote an assignment of value $d \in D_i$ to X_i by an ordered pair $\langle X_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* (PA) is a set of value assignments to variables, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists d \in D_i \wedge \langle X_i, d \rangle \in PA\}$. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if $X_{i_1}, X_{i_2}, \ldots, X_{i_k} \in vars(PA)$. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *full assignment* is a partial assignment that includes all the variables ($vars(PA) = \mathcal{X}$). A *solution* is a full assignment of minimal cost.

3 Standard Max-sum

The Max-Sum algorithm [3] operates on a *factor graph* which is a bipartite graph in which the nodes represent variables and constraints [1]. Each node representing a variable of the original DCOP is connected to all function-nodes that represent constraints

[1] We preserve the terminology of [3] and call constraint representing nodes in the factor graph "function nodes".

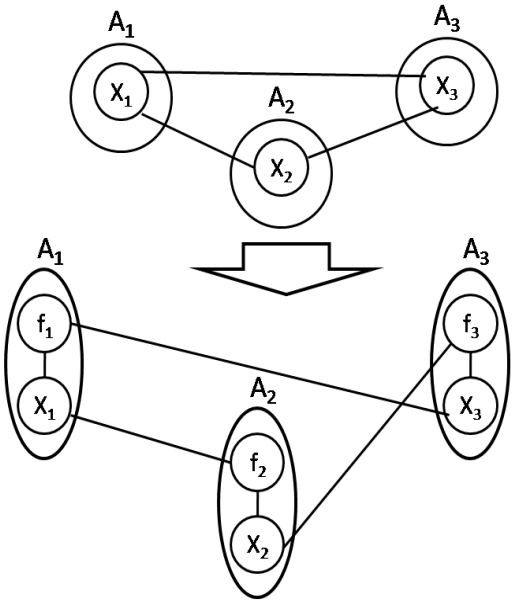


Fig. 1. Transformation of a DCOP to a factor graph

which it is involved in. Similarly, a function-node is connected to all variable-nodes that represent variables in the original DCOP which are included in the constraint it represents. Agents in Max-sum perform the roles of different nodes in the factor graph. We will assume that each agent takes the role of the variable-nodes which represent her own variables and for each function-node, one of the agents who's variable is involved in the constraint it represents, performs its role. Variable-nodes and function-nodes are considered as "agents" in Max-sum, i.e., they can send messages, read messages and perform computation.

Figure 1 demonstrates the transformation of a DCOP to a factor graph. On the top we have a DCOP with three agents, each holding a single variable. All variables are connected by binary constraints. On the bottom we have a factor graph. Each agent takes the role of the node representing her own variable and the role of one of the function-nodes representing a constraint it is involved in.

Figure 2 presents a sketch of the Max-sum algorithm. The code for variable-nodes and function-nodes is similar apart from the computation of the content of messages to be sent. For variable-nodes only data received from neighbors is considered. In messages sent by function-nodes the content is produced considering data received from neighbors and the original constraint represented by the function-node.

It remains to describe the process of the production of messages by the factor graph nodes. A message sent from a variable-node representing variable x to a function-node f at iteration i includes for each of the values $d \in D_x$, the sum of costs/utilities for this value she received from all function neighbors apart from f in iteration $i-1$. Formally,

Max-sum (node n)
1. $N_n \leftarrow$ all of n's neighboring nodes
2. **while** (no termination condition is met)
3. collect messages from N_n
4. **for each** $n' \in N_n$
5. **if** (n is a variable-node)
6. produce message $m_{n'}$ using messages from $N_n \setminus \{n'\}$
7. **if** (n is a function-node)
8. produce message $m_{n'}$ using constraint and messages from $N_n \setminus \{n'\}$
9. send $m_{n'}$ to n'

Fig. 2. Standard Max-sum.

for value $d \in D_x$ the message will include: $\sum_{f' \in F_x, f' \neq f} cost(f'.d)$, where F_x is the set of function-node neighbors of variable x and $cost(f'.d)$ is the cost/utility for value d included in the message received from f' in iteration $i-1$.

A message sent from a function-node f to a variable-node x in iteration i includes for each possible value $d \in D_x$ the best (minimal in a minimization problem, maximal in a maximization problem) cost/utility that can be achieved from any combination of assignments to the variables involved in f apart from x and the assignment of value d to variable x. Formally, in a minimization problem, the message from f to x includes for each value $d \in D_x$: $min_{ass_{-x}} cost(\langle x, d \rangle, ass_{-x})$, where ass_{-x} is a possible combination of assignments to variables involved in f not including x. The cost of an assignment $a = (\langle x, d \rangle, ass_{-x})$ is: $f(a) + \sum_{x' \in X_f, x' \neq x} cost(x'.d')$. Where $f(a)$ is the original cost in the constraint represented by f for the assignment a and $cost(x'.d')$ is the cost which was received in the message sent from node-variable x' in iteration $i-1$, for the value d' which is assigned to x' in a.

While the selection of value assignments to variables is not a part of the Max-sum algorithm, we need to describe how the solution is selected at the end of the run. Each variable selects the value assignment which received the best (lowest for a minimization problem and highest for a maximization problem) sum of costs/utilities included in the messages which were received most recently from its neighboring function-nodes. Formally, in a minimization problem, for variable x we select the value $\hat{d} \in D_x$ as follows: $\hat{d} = min_{d \in D_x} \sum_{f \in F_x} cost(f.d)$. Notice that the same information used by the variable-node to select the content of the messages it sends is used for selecting its assignment.

4 Max-sum with an Alternating DAG (Max-sum_AD)

In this section we propose a version of the Max-sum algorithm which guarantees convergence without eliminating constraints of the original DCOP. We will discuss exploration methods for this version of the algorithm in the next section.

In order to guarantee the convergence of the algorithm we need to avoid the pathology described in [3], caused by cycles of various sizes in the factor graph. To this end we select an order on all nodes in the factor graph. For example, we can order nodes

according to the indexes of agents performing their role in the algorithm. A node who's role is performed by agent A_i is ordered before a node who's role is performed by agent A_j if $i < j$. For variable and function nodes held by the same agent, we can determine (without loss of generality) that a variable-node is ordered before function-nodes held by the same agent (and not the other way around). Then, we perform the algorithm for l iterations allowing nodes to send messages only to nodes which are "after" them according to this order (in the case of ordering by indexes, send messages only to agents with larger indexes than their own). After l iterations in this direction, the order is reversed and messages are sent for the next l iterations only in the opposite direction (e.g., to agents with lower indexes) . In each direction the Max-sum algorithm is performed as described in Section3 with the exception of the restriction on the messages. Thus, in every calculation of a message sent by, for example, variable-node x to function-node f, all of the most recent messages x received from its neighboring functions $f' \in F_x$, $f' \neq f$ are considered. However, for neighbors which are before x according to the current order, the most recent messages were received following the previous iteration, while from neighboring function-nodes which are after x according to the current order, the last messages were received before the last alternation of directions.

The resulting algorithm Max-sum_AD has messages sent according to a directed acyclic graph (DAG) which is determined by the current order. Each time the order changes we get a DAG on which messages on each edge of the graph are sent only in a single direction.

Max-sum_AD (node n)

1. $o \leftarrow$ select an order on all nodes in the factor graph
2. $direct\_changes \leftarrow 0$
3. $N_n \leftarrow$ all of n's neighboring nodes
4. **while** (no termination condition is met)
5. **if** ($direct\_changes$ is even)
6. $current\_order \leftarrow o$
7. **else**
8. $current\_order \leftarrow reverse(o)$
9. $N_{prev\_n} \leftarrow \{\hat{n} \in N_n :$ $\hat{n}$ is before n in $current\_order\}$
10. $N_{follow\_n} \leftarrow N_n \setminus N_{prev\_n}$
11. **for**(k iterations)
12. collect messages from N_{prev_n}
13. **for each** $n' \in N_{follow\_n}$
14. **if** (n is a variable-node)
15. produce message m'_n using messages from $N_n \setminus \{n'\}$
16. **if** (n is a function-node)
17. produce message $m_{n'}$ using constraint and messages received from $N_n \setminus \{n'\}$
18. send $m_{n'}$ to n'
19. $direct\_changes \leftarrow direct\_changes + 1$

Fig. 3. Max-sum_AD.

Figure 3 presents a sketch of the Max-sum_AD algorithm. It deffer's from standard Max-sum in the selection of directions and the disjoint sets of neighbors from whom the nodes receive messages and to whom they send messages (lines 5 - 10). Keeping track of the number of direction changes allows us to determine the current direction and act accordingly (lines 2, 5 and 19).
Next we prove the convergence of Max-sum_AD.

Lemma 1 *For any node n in the factor graph, if l' is the longest path in the DAG from some other node to n, then after l' iterations in the same direction, the content of the messages n receives does not change until the next change of direction.*

Proof: We prove by induction on l'. For $l' = 0$, node n does not receive messages from any other node as long as the direction does not change. We assume the correction of the Lemma for any length l' of a path shorter than the longest path in the DAG, l. If we denote the last node in the path whose length is equal to l by n', then according to the assumption, all the neighbors that are sending messages to n' after $l - 1$ iterations receive messages with the same content in all the following iterations with the same $current\_order$. Thus, after $l - 1$ iterations the data they use to produce the content of the messages they send is fixed. Therefore, in the following iterations they will send the same messages to node n'. □.

An immediate corollary from Lemma 1 is that agents will not change their assignment selection for their variable after l iterations in the same direction until the direction is alternated, since the information used by variable-nodes for selecting an assignment is the same information they use for generating messages to function-nodes (see Section 3). Thus, the algorithm converges to a single complete assignment. The decision to escape it by changing direction is an algorithmic decision. Notice that after the first alternation of direction, although we send messages only in a single direction, the data passed in the last messages which were received before the change in direction is used for the calculation of the content of messages to be sent. Thus, all the constraints of the problem are considered.

5 Exploration Methods for Max-sum_AD

The Max-sum_AD algorithm presented above converges in linear time. After performing l iterations in each direction (where l as before is the length of the longest path in the DAG) we allow each of the constraints in the problem to be considered in the final selection of assignments, i.e., the algorithm is completely exploitive and converges to a solution after considering all the DCOP constraints . This process is deterministic. If after the second direction change we set all costs/utilities in the messages received most recently to zero, and perform l iterations according to the initial order and l in the reversed order, we will converge to the same solution. We will refer to this exploitive version of Max-sum_AD in which after every even change of directions we set all costs/utilities to zero as $plain$. Next, we propose exploration methods which can allow the Max-sum_AD algorithm to continue the search for a better solution.

1. In the first, instead of setting the costs/utilities to zero after even direction changes we continue to accumulate them as in standard Max-sum. We will refer to this method as $standard$.
2. The second method selects a random number of iterations to be performed in each direction. After each change of direction, a random number $1 \leq l' \leq k$ is selected uniformly and the algorithm is performed for l' iterations in one direction before a new l' is selected for the number of iterations to be performed in the reversed direction. We denote this method by *Random Number of Iterations Selection* (RNIS). The range k should allow convergence in some cases and avoid them in others. In our experiments we used $k = n$ where n was the number of nodes in the factor graph. There exists multiple methods for a random number selection in distributed systems (e.g., [1]). Specifically in Max-sum_AD we can have a single agent select the random numbers of iterations for future rounds and propagate this selection to all other agents via a BFS tree on the DAG as in the anytime framework proposed in [14].
3. The third method handles an unlucky selection of the order on the factor graph nodes which determines the DAG the algorithm uses. It selects a random order and performs the algorithm in both directions on this order before selecting an order again. We denote this method by *Random Order Selection* (ROS). In our implementation we selected an agent to be "first" in the order randomly and all the other agents were ordered according to their indexes following this agent (e.g., in a problem with 10 agents, if agent A_5 is selected to be first, the order is $A_5, A_6, ..., A10, A1, ..., A4$). We determined that for nodes which their role is performed by the same agent, variable-nodes come before function-nodes in the order.

6 Experimental Evaluation

We present a set of experiments that demonstrate the advantage of the proposed Max-sum_AD algorithm when combined with the proposed exploration methods, over the Max-sum algorithm.

The experiments were performed on minimization random DCOPs in which each agent holds a single variable. Each variable had five values in its domain. The network of constraints in each of the experiments, was generated randomly by selecting the probability p_1 of a constraint among any pair of agents/variables. The cost of any pair of assignments of values to a constrained pair of variables was selected uniformly between 1 and 10. Such uniform random DCOPs with constraint networks of n variables, k values in each domain, a constraint density of p_1 and a bounded range of costs/utilities are commonly used in experimental evaluations of centralized and distributed algorithms for solving constraint optimization problems [5, 4]. Other experimental evaluations of DCOPs include random max graph coloring problems [7, 13, 3], which are a subclass of random generated DCOPs.

Our experimental setup includes problems generated with 35 agents each. The factor graph generated for all versions of the Max-sum algorithm had agents performing the role of the variable-nodes representing their own variables, and for each constraint, we had the agent with the smaller index involved in it perform the role of the corresponding

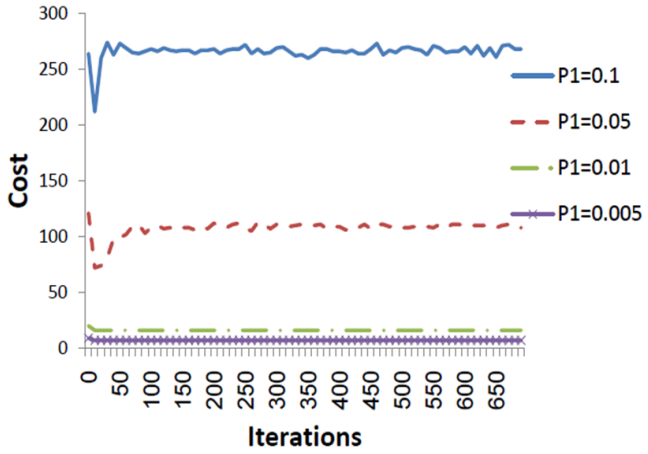


Fig. 4. Solution cost of Max-sum for every 10th iteration when solving problems with very low density

function-node. Figure 4 presents results for the standard Max-sum algorithm solving problems with very low density. Only for problems with extremely low density the algorithm converges. Thus, for the comparison of Max-sum with Max-sum_AD, we generated problems with two density parameters $p_1 = 0.1$ and $p_1 = 0.5$. For both of these density parameters Max-sum did not converge.

The Max-sum algorithm was compared with four versions of Max-sum_AD: $plain$, $standard$, $RNIS$ and ROS (see Section 5 for their description). We generated 50 random problems and ran the algorithms for 700 iterations on each of them. The results we present are an average on those 50 runs. To make sure that the Max-sum_AD algorithms converge we changed directions every 70 iterations (except in the RNIS version) which is the longest possible path in the DAG (in case the graph has a chain structure).

For each of the algorithms we present both the sum of the costs of constraints in the assignment it would have selected in each iteration and the *anytime value* (the best sum of costs found for some state visited up to this iteration). The framework proposed in [14] enhances DCOP local search algorithms with the *anytime* property. It uses a *Breadth First Search* (BFS) tree on the constraints graph in order to accumulate the costs of agents' states in the different steps during the execution of the algorithm. The anytime property in this framework is achieved with a very low overhead in time, memory and communication. In addition, it preserves a higher level of privacy than other DCOP algorithms which use tree structures [14].

Figure 5 presents for problems with constraint density $p_1 = 0.1$, for every ten'th iteration, the cost of the solution that would have been selected by the algorithm if the run would terminate at this iteration (we do not present the cost at each iteration to prevent the figure from being too dense). It is apparent that while Max-sum traverses states of low quality (with high costs) and the plain version of Max-sum_AD converges to the same solution over and over again, the versions of Max-sum_AD which are combined with exploration methods traverse lower cost states. The performance of $RNIS$ dete-

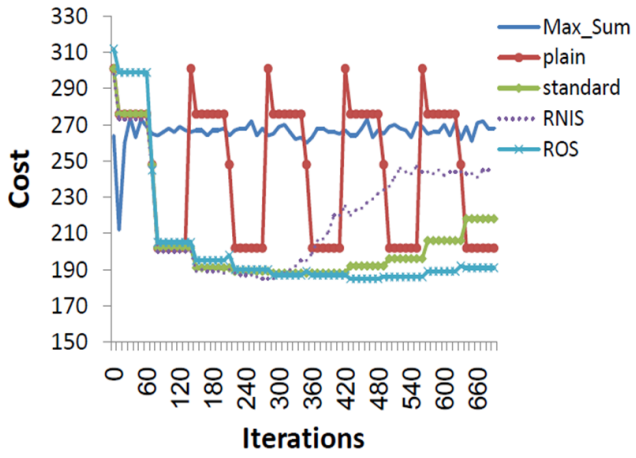


Fig. 5. Solution cost for every 10th iteration when solving problems with low density ($p_1 = 0.1$)

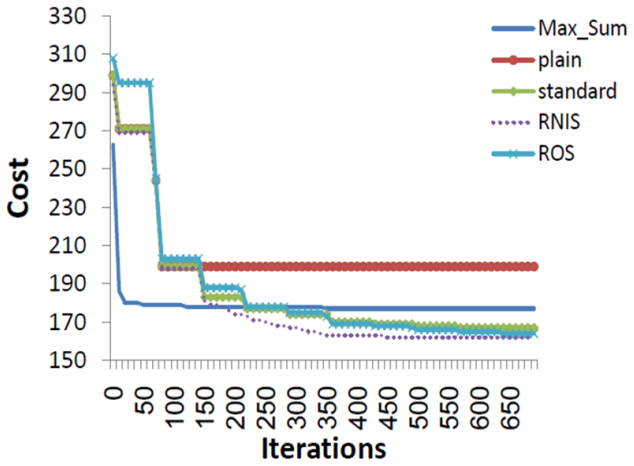


Fig. 6. Anytime cost for every 10th iteration when solving problems with low density ($p_1 = 0.1$)

riorates after the fourth change in direction and later the performance of the $standard$ heuristic deteriorates as well. On the other hand, the ROS heuristic continues to converge to high quality states with low costs. An interesting phenomenon to point out is that Max-sum visits relatively high quality states in the early iterations of the algorithm before its deterioration to an unsteady traverse of states with low quality. It seems that

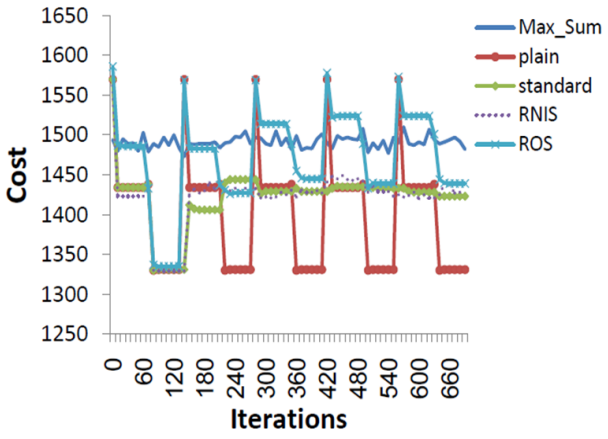


Fig. 7. Solution cost for every 10th iteration when solving problems with high density ($p_1 = 0.5$)

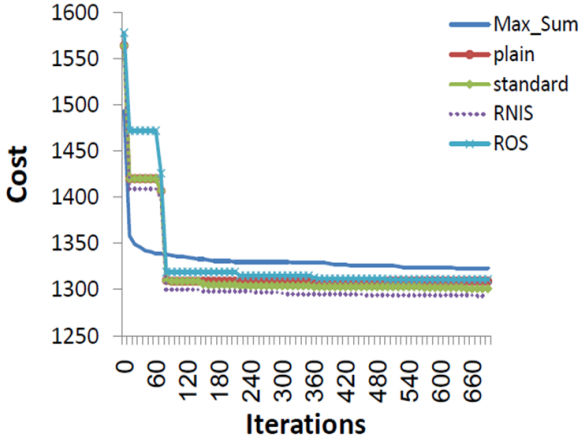


Fig. 8. Anytime cost for every 10th iteration when solving problems with high density ($p_1 = 0.5$)

it takes a number of iterations before the effect of the cycles on its performance begin. The anytime results for this experiment are presented in Figure 6. Notice that the anytime result selects the best among states in all the iterations and not only the ones which their cost was presented in Figure 5.

Similar results are presented in Figures 7 and 8 for problems with constraint density $p_1 = 0.5$. On dense problem the advantage of the different versions of Max-sum_AD over the standard Max-sum is more apparent. In addition the $plain$ version of the algorithm converges to a solution of high quality and the advantage achieved by the exploration methods is less apparent. Here, the exploitive performance which we observed

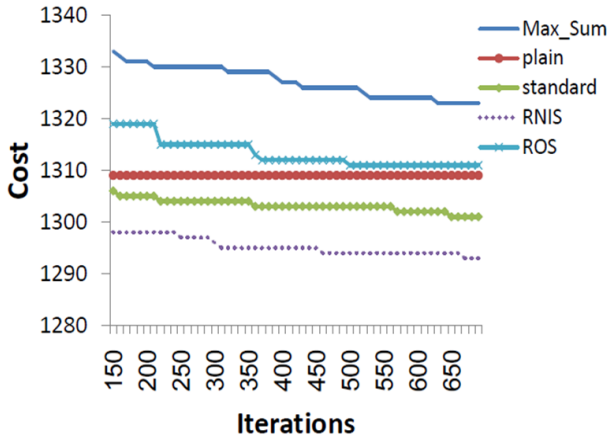


Fig. 9. A closer look at the anytime cost for every 10th iteration, starting after 140 iterations ($p_1 = 0.5$)

for Max-sum in the first iterations of Figure 5 does not appear. This is probably because the size of the cycles is smaller, thus their destructive effect is instantaneous. A closer look at the anytime result of the algorithms after the first 140 iterations is presented in Figure 9. While the ROS method does not improve on the $plain$ method, the standard method and RNIS find solutions with lower costs. All the versions of Max-sum_AD outperform Max-sum. It is important to notice that while the average cost in each iteration as presented in Figure 7 does not reveal an advantage of the exploration methods the anytime result does. This is because different iterations were successful when solving different problems. Thus, the average in each iteration does not reveal this success.

Figure 10 demonstrates the balance between exploitation and exploration of the proposed methods by presenting the states in the run of a single problem for ROS and RNIS. It is apparent that after each change in direction there is an exploration phase and a convergence to a solution. RNIS converges only when the random selected number of iterations in the same direction is large enough while ROS converges after each change in direction.

7 Conclusion

The Max-sum algorithm offers an innovative approach for solving DCOPs. Unfortunately, when problems include cycles of various sizes in the factor graph, the algorithm does not converge and the states it visits are of low quality.

In this paper we proposed a new version of the Max-sum algorithm, Max-sum_AD, which guarantees convergence. Max-sum_AD uses an alternating DAG to avoid cycles. We proved that the algorithm converges if the number of iterations it performs in a single direction is equal to or larger than the longest path in the DAG.

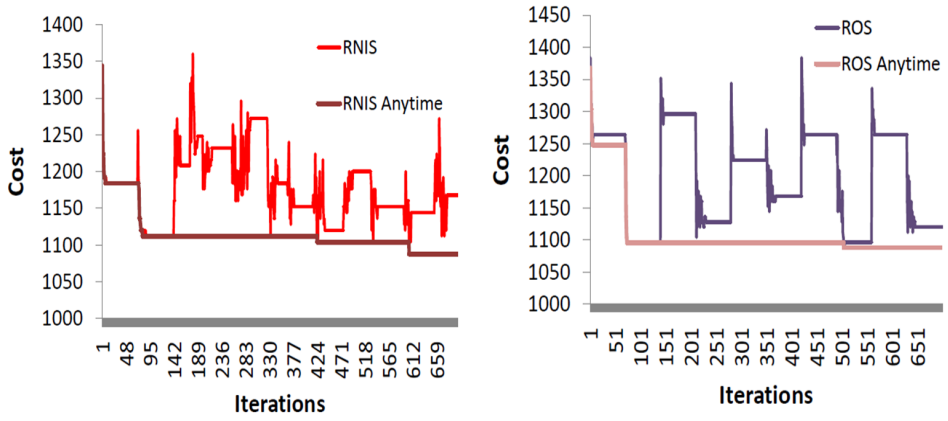


Fig. 10. A single run of RNIS (left) and ROS (right)

The guaranteed convergence of the strictly exploitive ("plain") algorithm serves as a baseline on which we add exploration elements and allow the algorithm to continue the search for a high quality solution. The use of the algorithm within the anytime framework proposed in [14] allows the selection of the best among the complete assignments it converges to as the algorithm's outcome.

Our empirical study reveals the advantage of the Max-sum_AD algorithm when combined with exploration methods over Max-sum. In the future we intend to investigate the compatibility of our exploration methods to realistic applications.

Acknowledgment: We thank Alessandro Farinelli for helping us understand the Max-sum algorithm.

References

1. B. Awerbuch and C. Scheideler. obust random number generation for peer-to-peer systems. *Principles of Distributed Systems, Lecture Notes in Computer Science*, 4305:275–289, 2006.
2. A. Farinelli, A. Rogers, and N. R. Jennings. Bounded approximate decentralised coordination using the max-sum algorithm. In *Proc. 13th Workshop on Distributed Constraint Reasoning (DCR) at IJCAI-09)*, pages 46–59, Pasadena, CA, July 2009.
3. A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, pages 639–646, 2008.
4. A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *J. of Artificial Intelligence Research*, 34:25–46, 2009.
5. J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.
6. R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *Proc. Parallel and Distributed Computing Systems PDCS)*, pages 432–439, September 2004.
7. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.

8. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
9. S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, 2010.
10. M. E. Taylor, M. Jain, Y. Jin, M. Yokoo, and M. Tambe. When should there be a "me" in "team"?: distributed multi-agent optimization under uncertainty. In *Proc. of the 9th conference on Autonomous Agents and Multi Agent Systems (AAMAS 2010)*, pages 109–116, May 2010.
11. W. T. L. Teacy, A. Farinelli, N. J. Grabham, P. Padhy, A. Rogers, and N. R. Jennings. Max-sum decentralised coordination for sensor systems. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1697–1698, 2008.
12. X. S. W. Yeoh and S. Koenig. Trading off solution quality for faster computation in dcop search algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 354–360, July 2009.
13. W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.
14. R. Zivan. Anytime local search for distributed constraint optimization. In *AAAI*, pages 393–398, Chicago, IL, USA, 2008.

Decentralised Parallel Machine Scheduling for Multi-Agent Task Allocation

Kathryn S. Macarthur[1], Meritxell Vinyals[2], Alessandro Farinelli[3], Sarvapali D. Ramchurn[4], and Nicholas R. Jennings[5]

{ksm08r, sdr, nrj}@ecs.soton.ac.uk[1,4,5], meritxell@iiia.csic.es[2], alessandro.farinelli@univr.it[3]

[1,4,5] School of Electronics and Computer Science, University of Southampton, UK
[2] IIIA, CSIC, Barcelona, Spain
[3] University of Verona, Italy

Abstract. Multi-agent task allocation problems pervade a wide range of real-world applications, such as search and rescue in disaster management, or grid computing. In these applications, where agents are given tasks to perform in parallel, it is often the case that the performance of all agents is judged based on the time taken by the slowest agent to complete its tasks. Hence, efficient distribution of tasks across heterogeneous agents is important to ensure a short completion time. An equivalent problem to this can be found in operations research, and is known as scheduling jobs on unrelated parallel machines (also known as $R||C_{\max}$). In this paper, we draw parallels between unrelated parallel machine scheduling and multi-agent task allocation problems, and, in so doing, we present the decentralised task distribution algorithm (DTDA), the first decentralised solution to $R||C_{\max}$. Empirical evaluation of the DTDA is shown to generate solutions within 86–97% of the optimal on sparse graphs, in the best case, whilst providing a very good estimate (within 1%) of the global solution at each agent.

1 Introduction

Multi-agent task allocation problems pervade a wide range of real-world scenarios, such as search and rescue in disaster management [10], and environmental monitoring with mobile robots [12]. In such problems, a set of agents, such as rescue agents in search and rescue, must work together to perform a set of tasks, often within a set amount of time. In particular, agents are given tasks to perform in parallel, and the performance of the team is usually judged based on the time taken by the slowest member of the team. For example, consider a team of firefighters that must put out fires in a building before medical personnel can enter and provide first aid for civilians — in this case, the medical personnel can enter the building only when all fires have been extinguished. In this case, the task allocation problem we focus on would consist of firefighter agents and firefighting tasks. Each firefighter would be assigned a number of fires to put out, and medical personnel would only be able to enter after the last fire has been

extinguished by the last firefighter. Thus, the performance of all the firefighters (i.e., how early the medical personnel can enter the building) is judged on how long the last firefighter has taken to finish. Hence, efficient distribution of tasks across such heterogeneous agents is important to ensure an early completion time. In more detail, in this context, agents must find a solution that ensures all tasks are performed in the shortest amount of time.

Now, an analogue to this particular class of task allocation problems has been widely studied in operations research and is known as *scheduling on unrelated parallel machines*, or $R||C_{\max}$ [4]. In this problem, there are a set of heterogeneous machines, and a set of tasks which must be performed by those machines (equivalent to agents), potentially under some constraints (for example, where a given machine cannot execute certain tasks), such that the maximum finish time across machines, known as the *makespan*, is minimised. However, while many algorithms (for example, [5,6,8], see [9] for a review) have been developed to solve $R||C_{\max}$, they all require the existence of some central authority. However, this introduces a central point of failure: for example, if communication to and from the central authority were to fail, then another authority would have to be found so that it could re-compute the solution and try to communicate it. In addition, in realistic large-scale environments, which can potentially have hundreds of agents, there is a need for an algorithm that will scale well in terms of communication and computation, which centralised algorithms are unable to do. Hence, the challenge we face is to build decentralised algorithms that are robust to failure, and so, there is a clear need for a multi-agent systems solution to solve $R||C_{\max}$ in our domains.

Against this background, in this paper, we develop a novel, decentralised, approximate algorithm to solve the unrelated parallel machine scheduling problem, called the Decentralised Task Distribution algorithm (DTDA). DTDA uses localised message passing through the min-max algorithm to find good quality, per-instance bounded approximate solutions in a distributed, efficient manner. In more detail, the min-max algorithm is a localised message passing algorithm from the GDL (Generalised Distributive Law) family [1], which factorises the global problem into local agent-level sub-problems, by exploiting possible independence among agents' actions. For example, assume two firefighters are in distant parts of a large building, and must decide which of the fires surrounding them they must put out. In this situation, the two firefighters can avoid considering each others' actions but should coordinate their own actions with any firefighters which are close by, and therefore would be making their decisions regarding some of the same fires.

This paper advances the state of the art in the following ways:

- First, we provide a novel representation of the $R||C_{\max}$ problem, in terms of a junction graph, which is a graphical model frequently used to represent factored functions [7].
- Second, we show how we can simplify the min-max algorithm, and then run it over this junction graph representation to generate approximate solutions to the $R||C_{\max}$ problem, with per-instance bounds, through decentralised

message passing between agents. To the best of our knowledge, this is the first known application of min-max to the $R||C_{\max}$ problem.
– Third, we empirically evaluate our approach, by comparing our performance with two benchmark algorithms (optimal and greedy), on graphs with differing average degrees. We find that it produces solutions within 86–97% of the optimal, in the best case, in sparse environments, and outperforms greedy by up to 16%.

The rest of this paper is structured as follows. In Section 2, we formulate $R||C_{\max}$. Next, we decompose the problem in Section 3 so that it can be distributed, and discuss how we simplify the min-max algorithm in Section 4. Then, we present our decentralised algorithm in Section 5. Next, we empirically evaluate the quality of the solutions given by the DTDA in Section 6. Finally, in Section 7, we conclude.

2 Problem Formulation

In this section, we formally describe the problem we introduced in Section 1. In more detail, our problem consists of finding an allocation of tasks to agents in order to to optimize overall execution performance in terms of the system's makespan. First, in Section 2.1, we provide the basic definitions of the environment. Then, in Section 2.2, we describe our objective function, which formalises the analogy with the $R||C_{\max}$ problem overviewed in Section 1.

2.1 Basic Definitions

Let the set of agents be denoted as $A = \{a_1, a_2, \ldots, a_{|A|}\}$, and the set of tasks to be completed as $T = \{t_1, t_2, \ldots, t_{|T|}\}$. Each agent a_i can perform a set of tasks $T_i \subseteq T$. For each agent $a_i \in A$ we denote a cost function, $\chi_i : T_i \to \mathbb{R}^+$, that returns the total run–time incurred at a_i to perform some task $t \in T_i$. Thus, $\chi_i(t_k)$ returns the application-specific runtime required for agent a_i to perform task t_k. A graphical representation of an example environment is given in Figure 1, in which there are 3 agents (circles) and 4 tasks (squares). Each agent is connected to the tasks it can potentially perform by lines in the graph, and edges are labelled with $\chi_i(t_k)$. Thus, for example, agent a_1 will incur a runtime of 30 to perform task t_2 whereas agent a_2 will only incur a runtime of 20.

Given this, the problem is to schedule all of the tasks in T across the agents in A such that all tasks are completed and the makespan is minimised. We formally define this objective in the next section.

2.2 Objective Function

To show the analogy with $R||C_{\max}$, consider the set of jobs as the set of agents' tasks and the set of machines as the set of agents. Specifically, the objective of $R||C_{\max}$ is to find a mapping $m : A \to 2^T$ from tasks to agents, such that the

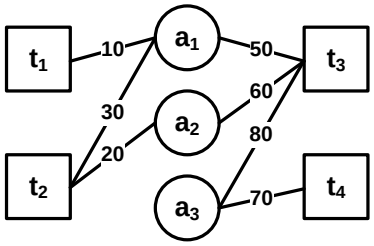


Fig. 1. A graphical representation of a sample $R||C_{\max}$ problem, in which agents are represented by circles, and tasks by squares. Edges between agents and tasks indicate an agent can perform a task, at a cost denoted on the edge.

makespan is minimized. In particular, we wish to find this mapping subject to a number of constraints. First, each task must only be computed by one agent:

$$m(a_i) \cap m(a_j) = \emptyset, \forall a_i, a_j \in A, i \neq j$$

and second, that all tasks must be computed:

$$\bigcup_{a_i \in A} m(a_i) = T$$

in which $m(a_i)$ denotes the set of tasks assigned to agent a_i, under mapping m. Given this, our objective is to find a mapping m^* as follows:

$$m^* = \arg \min_{m \in M} \max_{a_i \in A} \sum_{t_k \in m(a_i)} \chi_i(t_k) \tag{1}$$

where M is the set of all possible mappings. For instance, Figure 2 depicts an optimal mapping of the problem in Figure 1 where optimal assignments from agents to tasks are shown with arrows. Thus, the optimal mapping m^* is defined as: $m^*(a_1) = \{t_1, t_3\}$, $m^*(a_2) = \{t_2\}$ and $m^*(a_3) = \{t_4\}$ with a makespan value of $\max(10 + 50, 20, 70) = 70$.

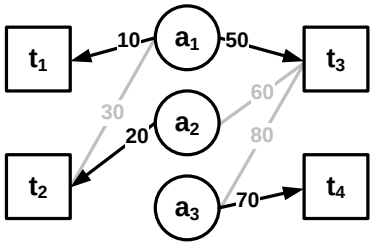


Fig. 2. An optimal mapping from agents to tasks, for the problem in Figure 1. Arrows between agents and tasks depict an agent being assigned to a task.

Now, in order to solve the objective function given in Equation (1) in a decentralised way, we first decompose the problem so that it can be modelled as a junction graph, like that shown in Figure 3.

3 R||C_{max} Representation

In more detail, a junction graph [7] is an undirected graph such that:

- each node i, known as a *clique*, represents a collection of variables, X_i.
- each clique node i in the junction graph has a potential, $\psi_i : X_i \rightarrow \mathbb{R}^+$, which is a function defined over the set of variables in the clique.
- two clique nodes i and j are joined by one edge that contains the intersection of the variables they represent.

Using this representation allows us to explicitly represent the interactions between agents, in terms of the common tasks they can complete. To do this, we represent each agent as a clique in the graph containing variables relating to the tasks that agent can complete. In so doing, the representation facilitates the application of a particular GDL [1] message-passing algorithm, called min-max, which we can use to find a solution to Equation (1) in a decentralised manner. We explain min-max in more detail in Section 4.

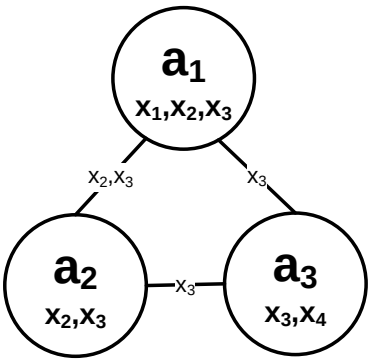


Fig. 3. The junction graph formulation of the scenario given in Figure 1. Large circles are cliques, with the elements of the cliques listed. Edges are labelled with common variables between cliques.

In order to apply min-max, we reformulate the objective function in Equation (1) in terms of a junction graph. Figure 3 depicts the junction graph representing the problem in Figure 1. We define the set of variables $X = \{x_1, \ldots, x_{|T|}\}$ to include one variable x_k for each task $t_k \in T$. Thus, the junction graph in Figure 3 contains four variables, $\{x_1, x_2, x_3, x_4\}$, which correspond to the four tasks in Figure 1. Each variable $x_k \in X$ takes a value from its domain, which contains

all of the IDs of agents that can complete task t_k. Hence, if $x_k = i$, then we know that agent a_i is allocated to t_k. For instance, the domain of x_2 in Figure 3 is composed of two values, 1 and 2, corresponding to the indices of the agents that can perform task 2, a_1 and a_2. Notice that, in doing this, we enforce the constraint that exactly one agent must perform every task.

Additionally, we use $X_i = \{x_k | t_k \in T_i\}$ as the set of variables representing the tasks agent a_i can perform. With slight abuse of notation, we use $\mathbf{X}_i$ to denote a configuration of the variables in X_i. Given this, in our formulation, an agent a_i's clique will contain all variables in X_i (in Figure 3, labels within circles denote agents' cliques). Thus, in Figure 3, the set of variables corresponding to agent a_2's clique, X_2, is composed of x_2 and x_3, which are the two tasks that a_2 can perform in Figure 1.

Finally, we encode the cost function of agent a_i as a potential function, $\psi_i(\mathbf{X}_i)$, representing the total time that a_i will take to compute the configuration $\mathbf{X}_i$. Formally:

$$\psi_i(\mathbf{X}_i) = \sum_{x_k \in \mathbf{X}_i, x_k = i} \chi_i(t_k) \tag{2}$$

Thus, in Figure 3 the potential function of agent a_2, ψ_2, which is defined over variables x_2 and x_3, returns a runtime of 60 for the configuration $x_2 = 1, x_3 = 2$, which is the runtime incurred at a_2 to complete task 3 in Figure 1.

By the definition of a junction graph, two agents, a_i and a_j, will be joined by an edge in the junction graph if and only if $X_i \cap X_j \neq \emptyset$. In Figure 3 edges are labelled with the intersection of two cliques. Thus, agent a_2 is linked to a_3 by an edge that contains the only common variable in their cliques: x_3. Given this, we denote agent a_i's neighbours, $\mathcal{N}(a_i)$, as the set of agents with which a_i shares at least one variable, and therefore, are neighbours in the junction graph.

Given all this, the junction graph encodes our objective function (Equation (1)) as follows:

$$\mathbf{X}^* = \arg\min_{\mathbf{X}} \max_{a_i \in A} \psi_i(\mathbf{X}_i) \tag{3}$$

where $\mathbf{X}_i$ is the *projection* of $\mathbf{X}$ over variables X_i. In more detail, given two sets of variables $X_i, X_j \subseteq X$, a projection of $\mathbf{X}$ over X_j contains the variable values found in $\mathbf{X}$ for all $x_k \in X_i \cap X_j$ in X_j .

Now that we have a junction graph formulation of the problem, we can decompose our objective function amongst the agents. In order to do this, we compute the marginal function $z_i(X_i)$ at each agent, which describes the dependency of the global objective function (given in Equation (1)) on agent a_i's clique variables. This is computed as follows:

$$z_i(\mathbf{X}_i) = \min_{\mathbf{X}_{-i}} \max_{a_j \in A} \psi_j(\mathbf{X}_j) \tag{4}$$

where $\mathbf{X}_{-i}$ is a configuration of X_{-i}, where $X_{-i} = X \setminus X_i$ and $\mathbf{X}_j$ is the projection of $\mathbf{X}_{-i}$ over the variables in X_j.

Finally, in the presence of a unique solution, the optimal state of a_i's clique variables is:

$$\mathbf{X}_i^* = \arg\min_{\mathbf{X}_i} z_i(\mathbf{X}_i) \tag{5}$$

This decomposition facilitates the application of the min-max GDL algorithm, as our operators here are min and max, to find a decentralised solution to $R||C_{\max}$. Thus, in the next section, we introduce min-max, and prove its most important property: that it will always converge within a finite number of iterations.

4 The min-max Algorithm

The min-max algorithm is a member of the GDL framework [1], which is a framework for localised message passing algorithms. We know from the literature (for example, [3]) that GDL algorithms are efficient (particularly in sparse graphs — in our case, where each agent can only perform a subset of the tasks), and provide generally good solutions, so it fits to apply one here. In addition, GDL algorithms are proven to converge to optimal solutions on acyclic graphs (which, in our case, would be junction trees). In more detail, GDL based algorithms are all based on a commutative semiring. Min-max is based on the commutative semiring $\langle \mathbb{R}^+, \min, \max, \infty, 0 \rangle$ where min is the additive operator and max the multiplicative operator.

Given a junction graph, the GDL approach consists of exchanging messages between agents and their neighbours in the junction graph. Let $X_{ij} = X_i \cap X_j$ be the intersection of variables of two neighbours, a_i and a_j, and $X_{i\setminus j} = X_i \setminus X_j$. In the GDL, agent a_i exchanges messages with a neighbour $a_j \in \mathcal{N}(a_i)$ containing the values of a function $\mu_{i\to j} : X_{ij} \to \mathbb{R}^+$.

Initially, all such functions are defined to be 0 (the semiring's multiplicative identity). Once messages have been received, the message is computed as follows:

$$\mu_{i\to j}(\mathbf{X}_{ij}) = \min_{\mathbf{X}_{i\setminus j}} \max \left[\psi_i(\mathbf{X}_i), \max_{a_k \in \mathcal{N}(a_i) | k \neq j} \mu_{k\to i}(\mathbf{X}_{ki}) \right] \tag{6}$$

where $\mathbf{X}_{ij}$ is a configuration of X_{ij}, $\mathbf{X}_{i\setminus j}$ is a configuration of $X_{i\setminus j}$, and $\mathbf{X}_i$ and $\mathbf{X}_{ki}$ stand for the projection of $\mathbf{X}_{ij}$,$\mathbf{X}_{i\setminus j}$ over variables in X_i and X_{ki} respectively.

Similarly, for each clique a_i, GDL computes an approximation of the marginal contribution of its variables, $\tilde{z}_i : \mathbf{X}_i \to \mathbb{R}^+$, as:

$$\tilde{z}_i(\mathbf{X}_i) = \max \left[\psi_i(\mathbf{X}_i), \max_{a_j \in \mathcal{N}(a_i)} \mu_{j\to i}(\mathbf{X}_{ji}) \right] \tag{7}$$

Now, the idempotency of max, the multiplicative operator [11], allows us to make a number of changes to the standard GDL formulation, which we explain below.

Idempotency implies that, for all $r \in \mathbb{R}^+$, $\max(r, r) = r$. Hence, the idempotency of the multiplicative operator implies that repeatedly combining the same information will not produce new information. Moreover, when an operator is idempotent, it defines a partial ordering over the set $\mathbb{R}^+$. In our case, both operations are idempotent. For the min operator, the order is the natural order of

real numbers: i.e., $r \leq s$ if and only if $\min(r, s) = r$. Meanwhile, for the max operator, the order is the inverse of the natural ordering of numbers: i.e., $r \geq s$ if and only if $\max(r, s) = r$. From these, we can deduce that, as min orders the elements in exact inverse to max, $\min(r, \max(r, s)) = r$.

Given all this, due to the idempotency of the min-max commutative semiring, the following equality holds for any $\mathbf{X}'_i$, $\mathbf{X}_i$ where $X'_i \subseteq X_i$:

$$\max \left[\psi_i(\mathbf{X}_i), \min_{\mathbf{X}'_i} \psi_i(\mathbf{X}'_i) \right] = \psi_i(\mathbf{X}_i) \tag{8}$$

This idempotency property is a feature we exploit in our implementation of min-max, to improve efficiency. In more detail, the idempotency of the min-max semiring, a property not shared with other non-idempotent GDL semirings, allows us to *simplify* the GDL equations such that:

- in Equation (6), when an agent a_i sends a message to a neighbour a_j, it does not need to marginalise out any previously received messages from a_j, thus reducing computation at agents.
- in Equation (7), the agent's marginal contributions can be computed recursively by combining messages from multiple iterations, which, again, reduces computation at the agent. This is because repeatedly combining messages from previous iterations will not change the approximate marginal contribution at an agent.

Thus, in the next section, we will introduce min-max based on these simplified equations, instead of the original GDL Equations ((6) and (7)), allowing us to simplify the computation at each agent when sending messages.

In addition to this, the idempotency property provides two further properties that make the min-max algorithm more efficient than non-idempotent GDL algorithms: (1) it is guaranteed to converge, even in cyclic graphs (Theorem 1); and (2) it provides an online per-instance bound on the optimal solution value of the problem that it approximates (which we will discuss later on, in Section 5.3). In what follows, we provide the formal proof of convergence.

Theorem 1. *The min-max algorithm is guaranteed to converge in a finite number of iterations.*

Proof. [2] prove the termination of idempotent commutative semirings (Theorem 8). Given the fact that min-max is an idempotent semiring, the min-max algorithm must terminate.

Now that we have shown why the min-max algorithm carries useful properties, in the next section, we present our decentralised task distribution algorithm (DTDA). Our algorithm consists of an algorithmic instantiation of the min-max algorithm, which, when combined with a value propagation phase, allows online per-instance bounds on solution quality to be obtained at each agent.

5 Decentralised Task Distribution Algorithm

Broadly speaking, the DTDA consists of two key steps: applying the min-max algorithm to compute an allocation of tasks to agents, and value propagation to ensure all agents choose the same assignment, and to compute the per-instance bound.

In more detail, the first step of the min-max algorithm propagates information across the agents to produce a set of solutions (we will explain how this can be a set later on). In the second phase (value propagation), agents are arranged in a tree structure, and one solution is chosen that is consistent with all other agents' solutions. We elaborate on these steps in what follows.

5.1 Applying min-max

In the first step of the DTDA, we apply the min-max algorithm over the junction graph described in Section 3, in order to find a set of solutions (distributions of tasks to agents).[1]

Algorithm 1 `minmax()` at agent a_i.

1: **procedure** `initialise`
2: Initialize messages $\mu_{i\to j}(\mathbf{X}_{ij}) = 0 \quad \forall j \in \mathcal{N}(i)$
3: $\tilde{z}_i(\mathbf{X}_i) = \psi_i(\mathbf{X}_i) = \sum_{x_k \in \mathbf{X}_i, x_k = i} \chi_i(t_k)$ `// Intialise marginal contribution`
4: Run procedure `send_messages`.
5:
6: **procedure** `received` $\mu_{j\to i}$
7: **if** $stored(j) \neq \mu_{j\to i}$ **then** `// received different message`
8: $stored(j) = \mu_{j\to i}$ `// update stored message`
9: Run procedure `update_marginal_contribution`
10: Run procedure `send_messages`.
11: **end if**
12:
13: **procedure** `send_messages`
14: **for** $j \in \mathcal{N}(i)$ **do**
15: Send $\mu_{i\to j}(\mathbf{X}_{ij})$ to a_j
16: **end for**
17:
18: **procedure** `find_solutions`
19: $\mathbf{X}_\mathbf{i}^* =$ all states with value $\min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i)$

We present the pseudocode for min-max in Algorithm 1. Now, an agent begins by running the procedure `initialise` (lines 1–4). Each agent starts by

[1] Note that we specify that a set of solutions is produced, because it is possible for more than one solution to have the same value. This is because the solution value is taken to be the largest makespan at one agent — therefore, many allocations of tasks and agents could give the same makespan.

initialising its stored outgoing messages to 0 (line 2), and its marginal contribution function, $\tilde{z}_i(\mathbf{X}_i)$, to the agent's potential, ψ_i, which encodes the agent's own cost function, computed as given in (2) (line 3).

After initialisation, each agent exchanges a message, $\mu_{i\to j}$, with each of its neighbours, $a_j \in \mathcal{N}(a_i)$, in order to find out their approximated marginal contribution for each configuration of the variables they share. This is done via the procedure `send_messages` (lines 13–16). The message $\mu_{i\to j}(X_{ij})$ is sent over all combinations of the variables in X_{ij} (i.e., the intersection of X_i and X_j). The content of the message from an agent a_i to a_j is, therefore, agent a_i's marginal contribution function:

$$\mu_{i\to j}(\mathbf{X}_{ij}) = \min_{\mathbf{X}_{i\setminus j}} \tilde{z}_i(\mathbf{X}_i) \tag{9}$$

When an agent a_i receives a message, it runs the procedure `received` (lines 6–11), in which the agent checks if the message it has received differs from the last message it received from that agent. This is important to ensure that the messages stop being sent when they stop changing, so the algorithm converges to a solution. If the message does differ (line 7), then the a_i updates its *stored* entry for the sending agent (line 8). Afterwards, the agent a_i runs the procedure `update_marginal_contribution` (line 9), which updates its marginal contribution values as follows:

$$\tilde{z}_i(\mathbf{X}_i) = \max\left\{\tilde{z}_i(\mathbf{X}_i), \max_{a_j\in\mathcal{N}(a_i)} \mu_{j\to i}(\mathbf{X}_{ji})\right\} \tag{10}$$

This marginal contribution is approximate because, as we said earlier, GDL algorithms can only compute exact solutions on acyclic graphs (i.e., a junction tree instead of a junction graph). Finally, agent a_i re-sends all of its messages (line 10) in case its marginal contribution value has changed (for example, if a new maximum $\mu_{j\to i}(\mathbf{X}_{ij})$ has been found).

These messages are passed amongst agents until their content no longer changes — at which point, the agent will ascertain the best states for its variables using the procedure `find_solutions` (lines 18–19). In more detail, this is done by assessing the configuration $\mathbf{X}_i^*$, with cost $\tilde{z}_i^*$, that minimises the agent's marginal contribution (line 19):

$$\mathbf{X}_i^* = \arg\min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i) \tag{11}$$

Hence, this equation provides an approximation of (3). We show in our empirical evaluation (in Section 6) that the solutions DTDA gives are of very good quality on a general problem. Next, we describe our value propagation phase which ensures that all agents apply the same solution and that computes the online per-instance bound of the approximate solution.

5.2 Value Propagation

Once the messages amongst agents have converged, and no further messages need to be sent, we introduce a two-part value propagation phase to ensure the

agents all set their values to produce a valid state, and are aware of the quality of their bound. This is required in part due to the likelihood of multiple solutions being present.

In the first part of this phase (see Algorithm 2, lines 1–10), we arrange the agents into a Depth-First Search (DFS) tree using distributed DFS. In particular, a DFS tree must always ensure that agents which are adjacent in the original graph are in the same branch. This ensures relative independence of nodes in different branches of the tree, allowing parallel search. For any given graph, our DFS tree is considered 'valid' if no ties (variable overlaps) between agents in different subtrees of the DFS tree exist. The outcome of this DFS is that each agent has set the values of its *parent* and *children* variables shown in Algorithm 2. Once this has occurred, the root node decides on a configuration of its variables to propagate, $\mathbf{X}_i^*$, as computed in (11), and sends this, along with $v_i = \psi_i(\mathbf{X}_i^*)$ (the actual value of the current solution) and $z_i(\mathbf{X}_i^*)$ (the value of the current solution as computed by min-max) to the node's children. Each of these children adds their own variables onto $\mathbf{X}_i^*$ (line 2), takes the maximum of v and z with what they have received (lines 3 and 4, respectively), and sends these new values onto their own children (line 5).

Algorithm 2 `valueprop` at agent a_i

Require: *parent*, *children*
1: **procedure** `received`$(\langle \mathbf{X}_p^*, v_p, \tilde{z}_p^* \rangle)$ from *parent*
2: $\mathbf{X}_i^* = \arg\min_{\mathbf{X}_{i \setminus p}} \tilde{z}_i(\mathbf{X}_{i \setminus p}; \mathbf{X}_p^*)$
3: $v_i = \max(v_p, \psi_i(\mathbf{X}_i^*))$
4: $\tilde{z}_i^* = \max(\tilde{z}_p^*, \tilde{z}_i(\mathbf{X}_i^*))$
5: Send $\langle \mathbf{X}_i^*, v_i, \tilde{z}_i^* \rangle$ to all $a_j \in$ *children*
6: **if** *children* $= \emptyset$ **then**
7: Send $\langle v_i, \tilde{z}_i^* \rangle$ to *parent*
8: **end if**
9:
10: **procedure** `received`$(\langle v_p, \tilde{z}_p^* \rangle)$ from *child*
11: **if** Received messages from all *child* $\in$ *children* **then**
12: $v_i = \max(v_p, \psi_i(\mathbf{X}_i^*))$
13: $\tilde{z}_i^* = \max(\tilde{z}_p^*, \tilde{z}_i(\mathbf{X}_i^*))$
14: $\rho = \tilde{z}_i^* / v_i$
15: Send $\langle v_i, \tilde{z}_i^* \rangle$ to *parent*.
16: **end if**

Once this first phase is complete (i.e., the messages have reached the leaf nodes), the leaf nodes pass their marginal contribution and makespan values (the actual value of the solution) back up the tree (lines 6 and 7), to ensure every agent can compute the quality of the solution. Then, when an agent has received such a message from each of its children (line 11), it will update its v and $z_i(\mathbf{X}_i^*)$ values (lines 12 and 13, respectively), calculate its approximation ratio ρ (line 14) — which is the agent's per-instance bound (this will be clarified

in the next section), and send the new v and $z_i(\mathbf{X}_i^*)$ values to its parent (line 15). Once these messages have reached the root of the tree, the DTDA is complete, all variables have values consistent across all agents, and all agents are aware of the quality of their solution.

5.3 Proving the Per-instance Bound

Next, we prove that the maximum value of the solution that minimises the approximate marginal contributions of the agents in min-max, or, more formally, $\tilde{z} = \max_{a_i \in A} \min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i)$, is a lower bound on the cost of the optimal solution of the $R||C_{\max}$ problem, as formulated in Equation (3) (Theorem 2). This lower bound can be used by agents to assess the quality of the min-max solution by bounding its error.

Before proving Theorem 2, we define the relation of *equivalence* among two functions.

Definition 1 (equivalence). *Given two functions $\alpha(X_\alpha)$ and $\beta(X_\beta)$ we say they are equivalent if they: (1) are defined over the same set of variables $X_\alpha = X_\beta$; and (2) return the same values for all possible configurations, $\alpha(\mathbf{X}) = \beta(\mathbf{X})$. We denote such relation of equivalence as $\alpha \equiv \beta$.*

Next, we prove two lemmas, that help to assess Theorem 2. First, in Lemma 1, we state that, at any iteration of the min-max algorithm, the function that results from the combination of the agents' marginal contributions, namely $\tilde{Z}(\mathbf{X}) = \max_{a_i \in A} \tilde{z}_i(\mathbf{X}_i)$, is *equivalent* to the objective function to minimise in $R||C_{\max}$, $\Psi(\mathbf{X}) = \max_{a_i \in A} \psi_i(\mathbf{X}_i)$. Thus, under Lemma 1, $\tilde{Z} \equiv \Psi$. Second, in Lemma 2, we state that $\tilde{z}$, defined as the maximum of the individual agents' marginal solutions, is a lower bound on the value of the optimal value of function $\tilde{Z}$.

We provide formal proofs for these two lemmas below.

Lemma 1. *At any iteration τ of the min-max algorithm, function $\tilde{Z}^\tau(\mathbf{X}) = \max_{a_i \in A} \tilde{z}_i^\tau(\mathbf{X}_i)$ is equivalent to the to the objective function to minimise in $R||C_{\max}$, $\Psi(\mathbf{X}) = \max_{a_i \in A} \Psi_i(\mathbf{X}_i)$.*

Proof. We prove this by induction on τ.

For $\tau = 0$ the case is trivial, $\tilde{Z}^0(\mathbf{X}) = \max_{a_i \in A} \tilde{z}_i^0(\mathbf{X}_i) = \max_{a_i \in A} \psi_i(\mathbf{X}_i) = \Psi(\mathbf{X})$.

Then we prove $\tau = n + 1$: that is, that $Z^{n+1} \equiv Z^n$, assuming that $\tau = n$ holds. $\tilde{Z}^{n+1}(\mathbf{X}) = \max_{a_i \in A} \max(\tilde{z}_i^n(X), \max_{a_j \in N(a_i)} \min_{\mathbf{X}_{j \setminus i}} \tilde{z}_j^n(\mathbf{X}_{j \setminus i}))$. Since the max operator is commutative and associative, $\tilde{Z}^{n+1}(\mathbf{X})$ can also be written as $\max_{a_i \in A} \max(\tilde{z}_i^n(\mathbf{X}_i), \max_{a_j \in \mathcal{N}(a_i)} \min_{\mathbf{X}_{i \setminus j}} \tilde{z}_i^n(\mathbf{X}_{i \setminus j}))$. Then, by exploiting the idempotency of the max operator (see Equation (8)), $\tilde{Z}^{n+1}(\mathbf{X})$ simplifies to $\max_{a_i \in A} \tilde{z}_i^n(\mathbf{X}_i)$ and $\tilde{Z}^{n+1} \equiv \tilde{Z}^n \equiv \Psi$. □

Lemma 2. *Given $\tilde{Z}(\mathbf{X}) = \max_{a_i \in A} \tilde{z}_i(\mathbf{X}_i)$, let $\tilde{z}^*$ be the value of the assignment x^* that minimises $\tilde{Z}(\mathbf{X})$. Then, $\tilde{z} = \max_{a_i \in A} \min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i)$ is a lower bound on $\tilde{z}^*$, $\tilde{z} \leq \tilde{z}^*$.*

Proof. We prove this by contradiction. Assume that there is an assignment $\mathbf{X}$ of X such that $\tilde{Z}(\mathbf{X}) \leq \max_{a_i \in A} \min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i)$. This leads to a contradiction, because it implies that at least one function $\tilde{z}_i$ evaluated at x is lower than its minimum, $\min_{\mathbf{X}_i} \tilde{z}_i(\mathbf{X}_i)$. □

Finally, we combine these two lemmas to prove our main result, in Theorem 2.

Theorem 2. *Let $\tilde{z}_i^\tau(\mathbf{X}_i)$ be agent a_i's marginal contribution function at iteration τ of the min-max algorithm. Then, $\tilde{z} = \max_{a_i \in A} \min_{\mathbf{X}_i} \tilde{z}_i^\tau(\mathbf{X}_i)$ is a lower bound on the value of the optimal solution, namely $\tilde{z} \leq \min_{\mathbf{X}} \Psi(\mathbf{X})$, where $\Psi(\mathbf{X}) = \max_{a_i \in A} \psi_i(\mathbf{X}_i)$.*

Proof. Since the optimal solution of two equivalent functions is the same, the result follows directly from Lemmas 1 and 2. □

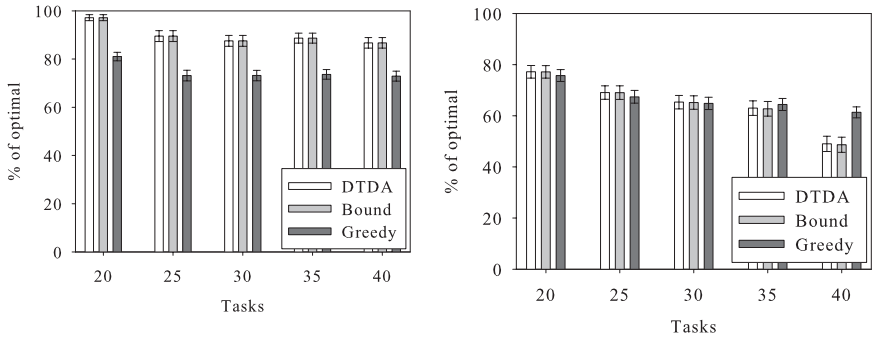
Therefore, under Theorem 2, at each iteration of the min-max algorithm, the maximum of the agents' marginal contributions, $\tilde{z} = \max_{a_i \in A} \min_{\mathbf{X}_i} \tilde{z}_i^\tau(\mathbf{X}_i)$, is a lower bound on the value of the optimal solution. Notice that, at each iteration, the agents' marginal contribution functions combine information from the messages using the max operator, so $\min_{\mathbf{X}_i} \tilde{z}_i^\tau(\mathbf{X}_i) \leq \min_{\mathbf{X}_i} \tilde{z}_i^{\tau+1}(\mathbf{X}_i)$. Therefore, the sequence of lower bounds is guaranteed to monotonically increase over iterations of min-max, thus providing a better approximation of the value of the optimal solution at each iteration. As shown in section 5.2, agents can, at the end of the min-max algorithm, assess this lower bound value to bound the error of the approximate solution found when running the min-max algorithm.

In the next section, we present our empirical evaluation of the DTDA. It is necessary for us to do this to show our algorithm finds good solutions, as the bound we provide on the quality of the approximations we give is per-instance, as opposed to an overall offline bound.

6 Empirical Evaluation

In this section, we compare the approximation found by the DTDA to a number of other algorithms, thus establishing the first decentralised benchmark for $R||C_{\max}$. Namely, we compare the DTDA against an optimal centralised algorithm, and a greedy algorithm. In more detail, the *optimal centralised algorithm (CA)* operates by solving a mixed integer program to find the optimal solution. We formulate the problem as a binary integer program, and then use IBM ILOG CPLEX[2] to find an optimal solution assigning tasks to agents. Next, in the *global greedy algorithm (Greedy)*, tasks are allocated to the agent that can complete them the fastest, and are considered in order of time required, from highest to lowest. In both these cases, we consider exactly the same problem the DTDA does — i.e., each agent can only perform a subset of the tasks. In addition, we plot the maximum bound found at an agent after executing the DTDA, ρ, as

[2] See `www.ibm.com/software/integration/optimization/cplex-optimizer/`



(a) $\sigma_t = 2$ (b) $\sigma_t = 3$

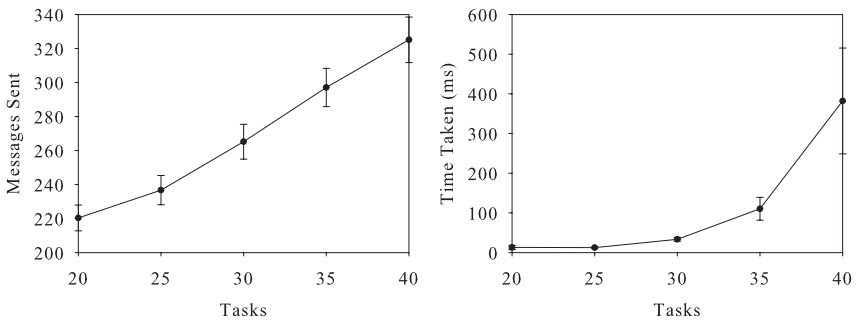
Fig. 4. Empirical Results: Utility gained for varying graph density.

computed in the value propagation phase, found in Section 5.2. Note that we do not compare to any existing approximate algorithms for $R||C_{\max}$ because, as we said earlier, there exist *no* decentralised algorithms for $R||C_{\max}$. Hence, our result establishes the first communication and computation benchmark for distributing the solution of $R||C_{\max}$ problems.

To evaluate the performance of DTDA, we plot the solutions obtained as a mean percentage of the optimal centralised solution, with error bars representing 95% confidence intervals in the mean. We calculate the mean approximation ratio of solutions obtained by each of these algorithms by dividing the achieved makespan by the optimal makespan (i.e., those obtained by CA), over 100 random scenarios, and use this to plot the percentage of the optimal obtained. In addition, we plot the mean total number of messages sent by DTDA, and the mean time taken to find a solution by DTDA.

We compare our algorithms in a number of average cases: specifically, sparse random graphs, and dense random graphs. In more detail, we generated 500 random scenarios with $|A| = 20$, $|T| = \{20, 25, 30, 25, 40\}$, and $\sigma_t \in \{2, 3\}$, where σ_t is the average degree of each task. The time taken for agent a_i to perform task t_j was calculated as $c_i \times c_j$, where $c_i \in \{1, \ldots, 100\}$ and $c_j \in \{0.1, \ldots, 1.1\}$, where c_i and c_j are both taken from uniform distributions. We present the utility results of these experiments in Figure 4, and the communication and computation results in Figure 5.

Figure 4(a) shows the performance of the DTDA versus greedy in a sparse environment, where each task can, on average, only be performed by two agents. Conversely, in Figure 4(b), we have the performance of DTDA versus greedy in a more dense environment, with an average of three agents being able to perform each task. The DTDA clearly outperforms greedy in the sparse graph by up to 16%; however, in the more dense graph, it is clear that the performance of DTDA does not warrant its application over greedy in this case. This shows that the DTDA is best applied on sparse graphs, as we intended, and is consistent with



(a) Messages Sent. (b) Computation Time.

Fig. 5. Empirical Results: Communication and Computation used where $\sigma_t = 2$.

other GDL algorithms [1]. Nevertheless, the DTDA's performance ranges from 97% of the optimal to 86% in the sparse graphs. In addition, the graphs show that the bound produced by the DTDA provides a very accurate estimation of the solution gained by the DTDA — so much so, that the two lines on the graph are barely distinguishable. Finally, in terms of communication and computation, Figure 5 (a) shows that the number of messages sent by DTDA increases almost linearly as the number of tasks increases. In contrast, Figure 5 (b) shows that the computation time increases exponentially in the number of tasks. Note that in Figure 5 we only plot results for $\sigma_t = 2$, as $\sigma_t = 3$ gave similar results.

7 Conclusions and Future Work

We have presented the first decentralised algorithm for finding solutions to the scheduling on unrelated parallel machines problem, known as $R||C_{\max}$. Our algorithm (DTDA) is also the first known application of the min-max algorithm to solve $R||C_{\max}$ in the literature. In addition, we are able to provide a per-instance bound on the quality of the solutions given, online. Empirically, we showed that the bound we find provides an accurate estimation of the global solution value, that the communication required by the DTDA scales linearly in the size of the environment, and that DTDA is able to find good quality solutions in environments which can be formulated as a sparse graph (from 97–86% of the optimal). In addition, we drew the parallel between $R||C_{\max}$ and multi-agent task allocation problems. However, we found that the DTDA holds no advantage over a greedy algorithm in more dense environments, partly because the state space explored at each agent in DTDA grows exponentially, and partly because the approximation given by using the min-max algorithm is not of high enough quality. While using the algorithm makes sense in task allocation environments where an agent only considers a limited number of tasks, the computation needed scales

exponentially in the size of the environment. Therefore, future work will focus on reducing the state space at each agent (e.g., by using techniques such as branch and bound), using spanning trees to improve solution quality on denser graphs, so that we can successfully apply DTDA to a wider range of problems, and evaluating DTDA's performance on other graph topologies, such as scale-free graphs.

References

1. Aji, S.M., McEliece, R.J.: The generalized distributive law. IEEE Transactions on Information Theory 46(2), 325–343 (2000)
2. Bistarelli, S., Gennari, R., Rossi, F.: Constraint propagation for soft constraints: Generalization and termination conditions. In: CP. pp. 83–97 (2000)
3. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proc. AAMAS–08. pp. 639–646 (2008)
4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnoy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326 (1979)
5. Horowitz, E., Sahni, S.: Exact and approximate algorithms for scheduling nonidentical processors. Journal of the ACM 23, 317–327 (April 1976)
6. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM 24(2), 280–289 (1977)
7. Jensen, F.V.: An Introduction to Bayesian Networks. Springer-Verlag (1996)
8. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical Programming 46, 259–271 (1990)
9. Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: A survey of milestones. Journal of the Operational Research Society 60, 41–68 (2009)
10. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralised Coordination in RobocupRescue. The Computer Journal 53(9), 1447–1461 (2010)
11. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
12. Stranders, R., Farinelli, A., Rogers, A., Jennings, N.: Decentralised coordination of mobile sensors using the max-sum algorithm. In: Proc. IJCAI–09. pp. 299–304 (2009)

Pseudo-tree-based Incomplete Algorithm for Distributed Constraint Optimization with Quality Bounds

Tenda Okimoto, Yongjoon Joe, Atsushi Iwasaki, and Makoto Yokoo

Kyushu University, Fukuoka 8190395, Japan
{tenda@agent., yongjoon@agent., iwasaki@, yokoo@}is.kyushu-u.ac.jp

Abstract. A Distributed Constraint Optimization Problem (DCOP) is a fundamental problem that can formalize various applications related to multi-agent cooperation. Since it is NP-hard, considering faster incomplete algorithms is necessary for large-scale applications. Most incomplete algorithms generally do not provide any guarantees on the quality of solutions. Some notable exceptions are DALO, the bounded max-sum algorithm, and ADPOP.

In this paper, we develop a new solution criterion called p-optimality and an incomplete algorithm for obtaining a p-optimal solution. The characteristics of this algorithm are as follows: (i) it can provide the upper bounds of the absolute/relative errors of the solution, which can be obtained a priori/a posteriori, respectively, (ii) it is based on a pseudo-tree, which is a widely used graph structure in complete DCOP algorithms, (iii) it is a one-shot type algorithm, which runs in polynomial-time in the number of agents n, and (iv) it has adjustable parameter p, so that agents can trade-off better solution quality against computational overhead. The evaluation results illustrate that this algorithm can obtain better quality solutions and bounds compared to existing bounded incomplete algorithms, while the run time of this algorithm is shorter.

1 Introduction

A Distributed Constraint Optimization Problem (DCOP) is a fundamental problem that can formalize various applications related to multi-agent cooperation. A DCOP consists of a set of agents, each of which needs to decide the value assignment of its variables so that the sum of the resulting rewards is maximized. Many application problems in multi-agent systems can be formalized as DCOPs, in particular, distributed resource allocation problems including distributed sensor networks [1] and meeting scheduling [2]. Various complete algorithms have been developed for finding globally optimal solution to DCOPs, e.g., DPOP [2], ADOPT [1], and OptAPO [3]. However, finding optimal DCOP solutions is NP-hard, so considering faster incomplete algorithms is necessary for large-scale applications. Various incomplete algorithms have been developed, e.g., DSA [4], MGM/DBA [5, 6], and ALS-DisCOP [7].

Most incomplete algorithms generally do not provide any guarantees on the quality of the solutions they compute. Notable exceptions are DALO [8], the bounded max-sum algorithm [9], and ADPOP [10]. Among these algorithms, DALO is unique since it can provide the bound of a solution a priori, i.e., the error bound is obtained before actually running the algorithm. Also, the obtained bound is independent of problem instances. On the other hand, the bounded max-sum algorithm and ADPOP can only provide the bound of a solution a posteriori, i.e., the error bound is obtained only after we actually run the algorithm and obtain an approximate solution. Having a priori bound is desirable, but a posteriori bound is usually more accurate.

In this paper, we develop an incomplete algorithm based on a new solution criterion called p-optimality. This algorithm can provide the upper bounds of the absolute/relative errors of the solution, which can be obtained a priori/a posteriori, respectively. These bounds are based on the induced width of a constraint graph and the maximal value of each reward function, but they are independent of problem instances. Induced width is a parameter that determines the complexity of many constraint optimization algorithms. This algorithm utilizes a graph structure called a pseudo-tree, which is widely used in complete DCOP algorithms such as ADOPT and DPOP. This algorithm can obtain an approximate solution with reasonable quality, while it is a one-shot type algorithm and runs in polynomial-time in the number of agents n. Thus, it is suitable for applications that need to obtain reasonable quality solutions (with quality guarantees) very quickly. Furthermore, in this algorithm, agents can adjust parameter p so that they can trade-off better solution quality against computational overhead.

DALO is an anytime algorithm based on the criteria of local optimality called k-size/t-distance optimality [8, 11] and has adjustable parameters k/t. Compared to this algorithm, our algorithm is a one-shot type algorithm, while DALO is an anytime algorithm, which repeatedly obtains new local optimal solutions until the deadline and returns the best solution obtained so far. Also, our algorithm can provide tighter bounds a priori. Furthermore, in our algorithm, the increase of computation/communication costs by increasing parameter p is more gradual compared to those for k-size/t-distance-optimality.

The bounded max-sum algorithm is a one-shot type algorithm. Compared to this algorithm, our algorithm has adjustable parameter p, while this algorithm has no adjustable parameter. Also, our algorithm can obtain a priori bound. Thus, agents can adjust parameter p before actually running the algorithm to obtain a solution with a desirable bound.

Our proposed algorithm is quite similar to ADPOP, which also eliminates edges among variables to bound the size of messages. ADPOP is also one-shot type algorithm and has an adjustable parameter. However, ADPOP uses a heuristic method to determine which edges to eliminate. As a result, it cannot obtain a priori bound. We can consider p-optimality gives a simple but theoretically well-founded method to determine which edges to eliminate in ADPOP.

Another advantage of our algorithm is that it can be used for a preprocessing phase before running a complete algorithm. Since our algorithm utilizes a pseudo-tree, it would be well-suited with pseudo-tree based complete algorithms.

The rest of this paper is organized as follows. Section 2 formalizes DCOP and provides basic terms related to the graphs. Section 3 introduces our incomplete algorithm and provides methods for estimating the error bound obtained by our algorithm. Section 4 evaluates the solution quality and the accuracy of the error bounds obtained by our algorithm. Section 5 concludes this paper.

2 Preliminaries

In this section, we briefly describe the formalization of Distributed Constraint Optimization Problems (DCOPs) and the basic terms for graphs.

Definition 1 (DCOP). *A distributed constraint optimization problem is defined by a set of agents S, a set of variables X, a set of binary constraint relations C, and a set of binary reward functions F. An agent i has its own variable x_i. A variable x_i takes its value from a finite, discrete domain D_i. A binary constraint relation (i, j) means there exists a constraint relation between x_i and x_j. For x_i and x_j, which have a constraint relation, the reward for an assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by a binary reward function $r_{i,j}(d_i, d_j) : D_i \times D_j \rightarrow \mathbb{R}$. For a value assignment to all variables A, let us denote*

$$R(A) = \sum_{(i,j)\in C, \{(x_i,d_i),(x_j,d_j)\}\subseteq A} r_{i,j}(d_i, d_j).$$

Then, an optimal assignment A^* is given as $\arg\max_A R(A)$, i.e., A^* is an assignment that maximizes the sum of the value of all reward functions.

Since there exists a one-to-one relationship between an agent and its variable, for notation simplicity, we occasionally don't distinguish an agent and its variable. For example, we define a constraint optimization problem by a tuple $\langle X, C, F\rangle$, where X is a set of agents/variables. In this paper, we assume all reward values are non-negative and that the maximal value of each binary reward function is bounded, i.e., we assume $\forall i, \forall j$, where $(i, j) \in C$, $\forall d_i \in D_i, \forall d_j \in D_j, 0 \leq r_{i,j}(d_i, d_j) \leq r_{max}$ holds.

A DCOP problem can be represented using a constraint graph, in which a node represents an agent/variable and an edge represents a constraint.

Definition 2 (Constraint Graph). *For a distributed constraint optimization problem $\langle X, C, F\rangle$, we say $G = (X, C)$ as a constraint graph of $\langle X, C, F\rangle$. More specifically, a constraint graph is obtained by assuming each agent/variable as a node, and each binary constraint relation as an edge.*

In this paper, we consider subgraphs that are obtained by removing several edges/constraints from the original DCOP/constraint graph. We define a reward obtained in a subgraph as follows.

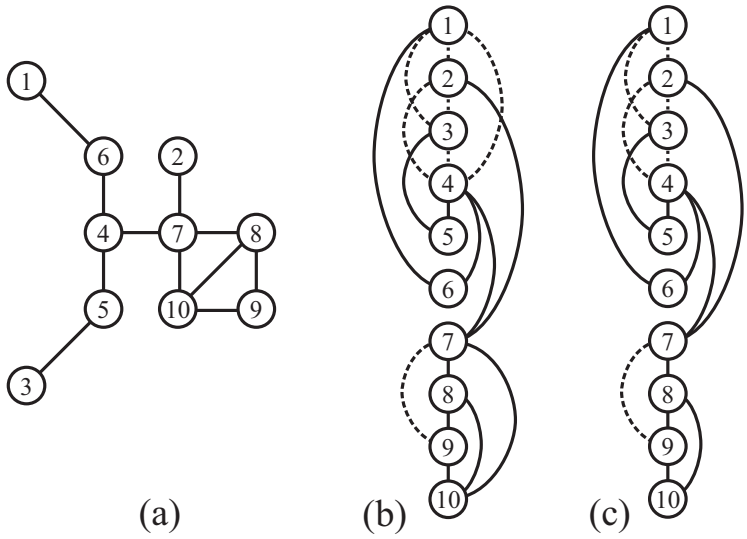


Fig. 1. (a) shows a constraint graph with ten nodes. (b) shows the induced chordal graph of (a) based on $o = 1 \prec \ldots \prec 10$. Induced width of (b) is three. (c) shows the subgraph of (b) obtained by removing edges $(1, 4)$ and $(7, 10)$. This graph is not chordal.

Definition 3 (Rewards in a subgraph). *For a distributed constraint optimization problem $\langle X, C, F\rangle$, its constraint graph $G = (X, C)$, and $G' = (X, C')$, which is a subgraph of G, i.e., $C' \subseteq C$, we define the rewards of an assignment A in subgraph $G' = (X, C')$ (denoted as $R_{C'}(A)$) as*

$$R_{C'}(A) = \sum_{(i,j)\in C', \{(x_i,d_i),(x_j,d_j)\}\subseteq A} r_{i,j}(d_i, d_j).$$

Let us introduce several basic terms for graphs.

Definition 4 (Undirected graph). *An undirected graph $G = (V, E)$ consists of a set of nodes $V = \{1, \ldots, n\}$ and a set of edges between nodes E. We denote $e \in E$ using nodes connected by the edge as $e = (i, j)$.*

Definition 5 (Connected graph). *We say an undirected graph $G = (V, E)$ is connected if any two nodes $i, j \in V$ are reachable via edges in E.*

In the rest of this paper, we assume a graph is connected.

Definition 6 (Neighboring nodes). *For a graph $G = (V, E)$ and a node $i \in V$, we call $Nb(E, i) = \{j \mid (i, j) \in E\}$ as i's neighboring nodes.*

Definition 7 (Total ordering among nodes). *A total ordering among nodes o is a permutation of a sequence of nodes $\langle 1, 2, \ldots, n\rangle$. We say node i precedes*

node j (denoted as $i \prec j$), if i occurs before j in o. We also denote ord(i) for the i-th node in a total ordering o.

Definition 8 (Ancestors). *For a graph $G = (V, E)$, a total ordering o, and a node $i \in V$, we call $A(E, o, i) = \{j \mid (i, j) \in E \ \wedge \ j \prec i\}$ as i's ancestors.*

Definition 9 (Chordal graph based on total ordering). *For a graph $G = (V, E)$ and a total ordering o, we say G is a chordal graph based on total ordering o when the following condition holds:*

- *$\forall i, \forall j, \forall k \in V$, if $j, k \in A(E, o, i)$, then $(j, k) \in E$.*

Definition 10 (Induced chordal graph based on total ordering). *For a graph $G = (V, E)$ and a total ordering o, we say a chordal graph $G' = (V, E')$ based on total ordering o, which is obtained by the following procedure, as an induced chordal graph [1] of G based on total ordering o.*

1. *Set E' to E.*
2. *Choose each node $i \in V$ from the last to the first based on o and apply the following procedure.*
   - *if $\exists j, \exists k \in A(E', o, i)$ s.t. $(j, k) \notin E'$, then set E' to $E' \cup \{(j, k)\}$.*
3. *Return $G' = (V, E')$.*

Next, we introduce a parameter called *induced width*, which can be used as a measure for checking how close a given graph is to a tree. For example, if the induced width of a graph is one, it is a tree. Also, the induced width of a complete graph with n variables is $n - 1$.

Definition 11 (Width based on total ordering). *For a graph $G = (V, E)$, a total ordering o, and a node $i \in V$, we call $|A(E, o, i)|$ as the width of node i based on total ordering o. Furthermore, we call $\max_{i \in V} |A(E, o, i)|$ as the width of graph G based on total ordering o and is denoted as $w(G, o)$.*

Definition 12 (Induced width based on total ordering). *For a graph $G = (V, E)$ and a total ordering o, we call $w(G', o)$ as the induced width of G based on total ordering o, where $G' = (V, E')$ is the induced chordal graph of G based on total ordering o.*

We show a simple example of the induced chordal graph of a graph and its induced width.

Example 1 (Induced width of induced chordal graph). Figure 1-(a) shows a constraint graph with ten nodes. (b) presents the induced chordal graph based on total ordering $o = 1 \prec \ldots \prec 10$. The ancestors of node 10 are nodes 7, 8, and 9. Since no edge exists between ancestors 7 and 9, edge $(7, 9)$ is added. Similarly, several new edges are added (shown as broken lines). The induced width of (b) is three.

[1] In constraint reasoning literature [12], such a graph is simply called an *induced* graph. However, the term *induced* is used in a more general meaning in graph theory. Thus, we use a more specific term, i.e., *induced chordal graph* in this paper.

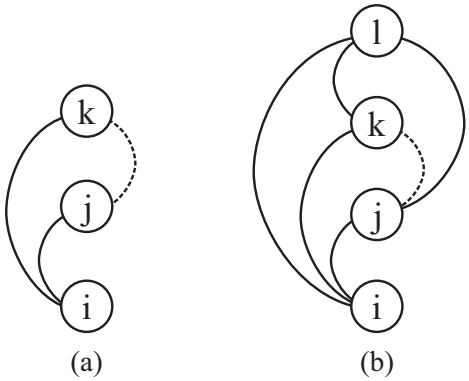


Fig. 2. (a) presents a part of p-reduced graph $G'' = (V, E'')$, where $j, k \in A(E'', o, i)$, and $(j, k) \notin E''$. (b) presents a situation where (j, k) is not j's first back-edge in G, i.e., there exists node l s.t. $l \prec k$, $(j, l) \in E$, and $(j, l) \notin E''$.

A pseudo-tree is a special graph structure, where a unique root node exists and each non-root node has a parent node.

Definition 13 (Pseudo-tree representation of chordal graph based on total ordering). *A chordal graph $G = (V, E)$ based on total ordering o can be assumed as a pseudo-tree as follows: (i) the node that appears first in o is the root node, and (ii) for each non-root node i, i's parent is node j, where $j \in A(E, o, i)$ and $\forall k \in A(E, o, i)$ and $k \neq j$, $k \prec j$ holds.*

Definition 14 (Back-edge). *When assuming a chordal graph $G = (V, E)$ based on total ordering o as a pseudo-tree, we say an edge (i, j) is a back-edge of i, if $j \in A(E, o, i)$ and j is not i's parent. Also, when $(i, j_1), (i, j_2), \ldots, (i, j_k)$ are all back-edges of i, and $j_1 \prec j_2 \prec \ldots \prec j_k$ holds, we call $(i, j_1), (i, j_2), \ldots,$ (i, j_k) as first back-edge, second back-edge, ..., k-th back-edge, respectively.*

Clearly, a node has at most $w(G, o) - 1$ back-edges.

3 Bounded Incomplete Algorithm based on Induced Width

In this section, we describe our new incomplete algorithm based on the induced width of a constraint graph. The basic idea of this algorithm is that we remove several edges from a constraint graph, so that the induced width of the remaining graph is bounded. Then we compute the optimal solution of the remaining graph, which is used as the approximate solution of the original graph.

3.1 Incomplete Algorithm and p-optimality

Our proposed incomplete algorithm has two phases:

Phase 1: Generate a subgraph from the induced chordal graph based on the total ordering by removing several edges, so that the induced width of the induced chordal graph obtained from the subgraph is bounded by parameter p.

Phase 2: Find an optimal solution to the graph obtained in Phase 1 using any complete DCOP algorithms.

First, let us describe Phase 1. Our goal is to obtain a subgraph so that the induced width of the induced chordal graph obtained from the subgraph equals p. At the same time, we want to bound the number of removed edges. This is not easy. One might imagine that we can easily obtain such a subgraph by just removing the back-edges so that all nodes have at most $p-1$ back-edges. However, by this simple method, we cannot guarantee that the remaining graph is chordal and we might need to add some edges to make it chordal. As a result, the induced width of the induced chordal graph can be more than p.

Let us show an example where the simple method does not work.

Example 2 (Simple method does not work). Figure 1-(c) presents the subgraph of (b) in Example 1. If we simply remove edges $(1,4)$ and $(7,10)$, each node has at most two edges with its ancestors (in (c)). However, the graph shown in (c) is not chordal, i.e., edge $(1,4)$ is missing, while there exist edges $(1,6)$ and $(4,6)$.

We develop a method for Phase 1 as follows. We call the obtained subgraph a *p-reduced graph*.

Definition 15 (p-reduced graph). *For a induced chordal graph $G=(V,E)$ based on total ordering o, we say a graph $G'=(V,E')$ obtained by the following procedure as p-reduced graph of G (where $1 \leq p \leq w(G,o)$):*

1. *Set E' to E.*
2. *Repeat the following procedure $w(G,o)-p$ times*
   - *For each $i \in V$ where $p+1 \leq ord(i) \leq w(G,o)$ remove the first back-edge in $G'=(V,E')$ from E' if there is one.*
3. *Return $G'=(V,E')$.*

Assuming that the agents know the pseudo-tree among them, running this procedure by these agents is quite simple. For obtaining the p-reduced graph, each agent i ($p+1 \leq ord(i) \leq w(G,o)$) simply removes its first back-edge, second back-edge, ..., $(w(G,o)-p)$-th back-edge.

Theorem 1. *For a induced chordal graph $G=(V,E)$ based on total ordering o, for any $1 \leq p \leq w(G,o)$, and G's p-reduced graph $G'=(V,E')$, the following conditions hold:*

1. *G' is a chordal graph based on total ordering o.*

2. $w(G', o)$ *is* p*.*

Proof. When obtaining p-reduced graph G', for each node i ($p+1 \leq ord(i) \leq w(G,o)$), its first back-edge is repeatedly removed $w(G,o) - p$ times. Since the number of back-edges is at most $w(G,o)-1$, the number of remaining back-edges is at most $p-1$. Also, there exists at least one node who has exactly $w(G,o)-1$ back-edges. Thus, since the remaining back-edges for the node are $p-1$, $w(G', o)$, i.e., the width of G' based on o, is p.

Next, we show that G' is a chordal graph based on total ordering o. Since p-reduced graph G' is obtained by repeatedly removing first back-edges for each node, it suffices to show that graph $G'' = (V, E'')$, which is obtained by removing first back-edges for each node in G, is a chordal graph based on total ordering o. We prove this fact by contradiction, i.e., we derive a contradiction by assuming that $\exists i \in V, \exists j, \exists k \in A(E'', o, i)$, s.t., $(j,k) \notin E''$. Without loss of generality, we can assume $k \prec j$ (Fig. 2-(a)).

Since $G = (V, E)$ is a chordal graph based on total ordering o, $(j,k) \in E$ holds. Furthermore, since $(j,k) \notin E''$, (j,k) must be the first back-edge of j in G. Also, since $k \in A(E'', o, i)$, $(i,k) \in E''$ holds. Thus, there exists node l s.t. $l \prec k$, $(i,l) \in E$, and $(i,l) \notin E''$ holds, i.e., (i,l) is i's first back-edge in G and is removed in G''. Furthermore, since $G = (V, E)$ is a chordal graph based on total ordering o, and $(i,l) \in E$ and $(i,j) \in E$ hold, $(j,l) \in E$ must hold (Fig. 2-(b)). However, since $l \prec k$, (j,k) cannot be j's first back-edge in G. This is a contradiction. Thus, $G'' = (V, E'')$ must be a chordal graph based on total ordering o.

We introduce a new criterion for approximated solutions.

Definition 16 (p-optimality). *We say an assignment A is p-optimal for a distributed constraint optimization problem $\langle X, C, F \rangle$ and a total ordering o, when A maximizes the total rewards in $G'' = (X, C'')$, where $G' = (X, C')$ is an induced chordal graph of $G = (X, C)$ based on total ordering o, and $G'' = (X, C'')$ is the p-reduced graph of G'. More specifically, $\forall A', R_{C''}(A) \geq R_{C''}(A')$ holds.*

Next, let us describe Phase 2. To find a p-optimal solution, we can use any complete DCOP algorithms. We use the obtained p-optimal solution as an approximate solution of the original graph. In particular, since we already obtained a pseudo-tree whose induced width is bounded, using pseudo-tree-based DCOP algorithms would be convenient.

3.2 Quality Guarantees

We provide two methods for estimating the error of the solution obtained by our algorithm. One method estimates absolute error which can be obtained a priori, i.e., agents can obtain the estimate before obtaining an approximate solution. Thus, agents can choose parameter p based on the estimation before actually obtaining an approximate solution.

Theorem 2. *For a distributed constraint optimization problem $\langle X, C, F\rangle$, its constraint graph $G = (X, C)$, and a total ordering o, if A is p-optimal, then the following condition holds among $R(A^*)$ and $R(A)$, where A^* is an optimal assignment:*

$$R(A^*) - R(A) \leq r_{max} \times \sum_{k=1}^{w(G,o)-p} (|X| - (k+1))$$

Proof. Since A is p-optimal, for $G' = (X, C')$, which is an induced chordal graph of G based on total ordering o, and $G'' = (X, C'')$, which is a p-reduced graph of G', the following condition holds:

$$R_{C''}(A^*) \leq R_{C''}(A).$$

Furthermore, C' is obtained by adding edges to C, and C'' is obtained from C' by removing at most $\sum_{k=1}^{w(G,o)-p}(|X| - (k+1))$ edges. Since the maximal reward of each edge is bounded by r_{max}, the following condition holds:

$$R(A^*) \leq R_{C''}(A^*) + r_{max} \times \sum_{k=1}^{w(G,o)-p} (|X| - (k+1)).$$

Furthermore, it is clear that the following condition holds:

$$R_{C''}(A) \leq R(A).$$

Thus, we obtain

$$R(A^*) - R(A) \leq r_{max} \times \sum_{k=1}^{w(G,o)-p} (|X| - (k+1)).$$

Intuitively, the absolute error is given by the product of r_{max} and the maximal number of removed back-edges.

Furthermore, we can compute the upper bound of the relative error using a method similar to ADPOP [10]. Note that this error bound can be obtained only a posteriori, i.e., we first need to obtain an approximate solution, then, we know the upper-bound of the relative error. Intuitively, if we remove a back-edge connecting i and j, we add an edge that connects i and j', where j' is a copy of j but it is connected only to i and has no unary reward. If we add an equality constraint between j and j', this problem is equivalent to the original problem. By ignoring such a constraint, we obtain a relaxed problem. Note that the induced width of this relaxed problem is p. This method, which ignores some dependencies among variables, is similar to minibucket elimination scheme [12].

4 Experimental Evaluation

In this section, we evaluate the solution quality and the accuracy of the error bounds obtained by our algorithm and show comparisons with DALO-t [13] and

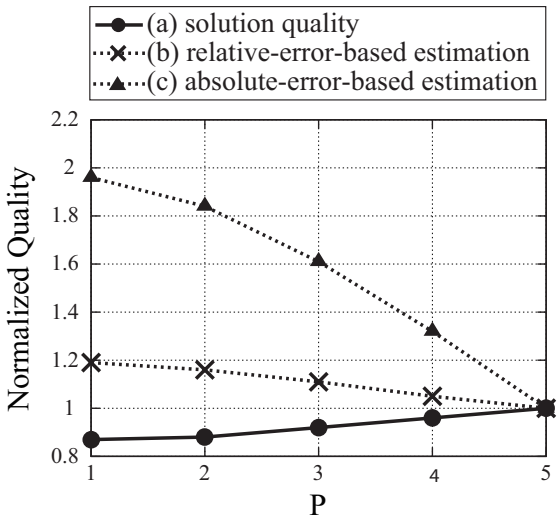


Fig. 3. (a), (b), and (c) in p-optimal algorithm for graphs with 20 nodes, induced width 5, and density 0.4. Value closer to 1 is desirable.

the bounded max-sum algorithm [9]. In our evaluations, we use the following problem instances. The domain size of each variable is three, and the reward of each binary constraint is in the range [0,. . . ,99]. Each data point in a graph represents an average of 30 problem instances. We generate random graphs with a fixed induced width. For Phase 2 of our p-optimal algorithm, we use the DPOP algorithm with FRODO [14](version 2.7.1). For comparison, we use the DALO-t algorithm that obtains t-distance-optimal solutions, since [8] shows that the error bounds for t-distance-optimality are usually better than that for k-size optimality. In our comparison, we mostly use settings p=1 and t=1.

First, we show (a) the quality of an obtained solution, (b) the estimated quality of an optimal solution based on the relative error bound, and (c) the estimated quality of an optimal solution based on the absolute error bound for the p-optimal algorithm. The results of (a), (b), and (c) are normalized by the quality of an actual optimal solution, where (a) should be less than 1, and (b) and (c) should be more than 1. For all of them, a value closer to 1 is desirable. Figure 3 shows these values for graphs with 20 nodes, induced width 5, and the density of the binary constraints 0.4. We vary parameter p from 1 to 5. Note that when we set the number of nodes to 20 and the induced width to 5, we cannot create a graph whose density is greater than 0.6. We can see that the obtained solution quality and estimation are reasonable for most cases, except that (c) becomes rather inaccurate when $p = 1$. This is because the number of removed edges is large. In such a case, we need to increase p to obtain a better estimation.

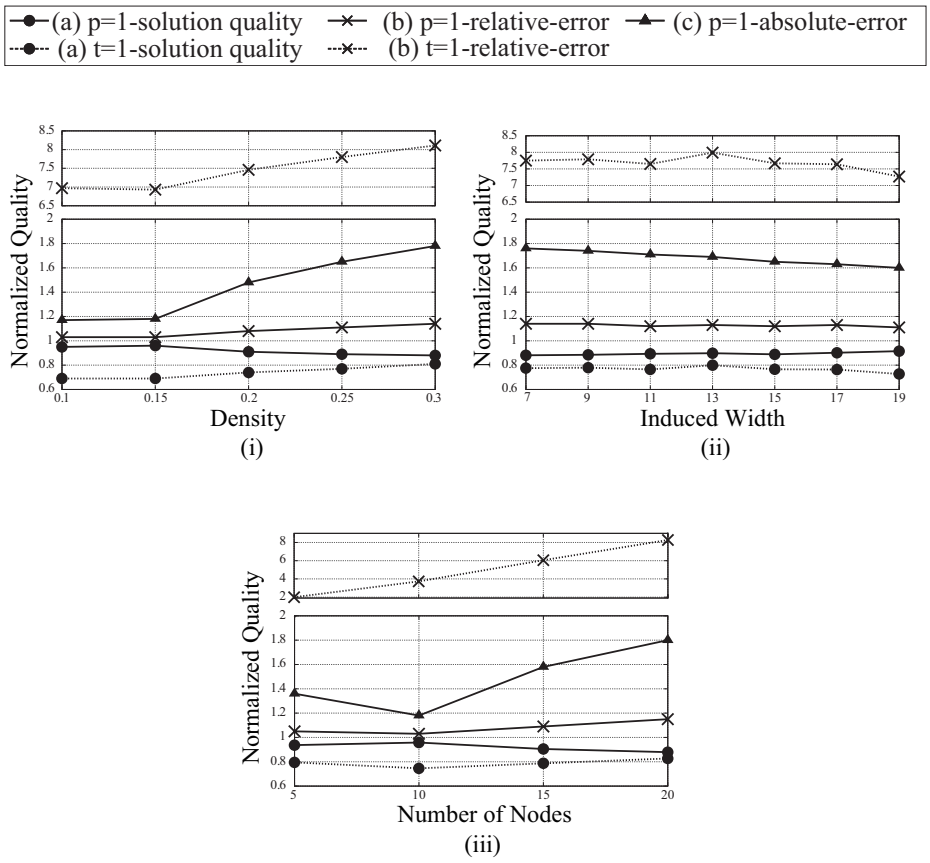


Fig. 4. (a), (b), and (c) in p=1-optimal algorithm and DALO-t=1 for graphs with (i) 20 nodes, induced width 5, (ii) 20 nodes, density 0.3, and (iii) density 0.3, induced width 3. Broken line indicates results for DALO-t=1. Value closer to 1 is desirable.

Next, we compare our algorithm for p=1-optimality and DALO-t=1 for t=1-distance-optimality. Usually, DALO-t is used as an anytime algorithm, i.e., it continuously obtains t-optimal solutions. In this paper, we stop DALO-t when the first t-optimal solution is found. Figure 4(i) shows (a), (b), and (c) in the p=1-optimal algorithm and in DALO-t=1 for graphs with 20 nodes and induced width 5, varying the density. A value closer to 1 is desirable. The broken lines indicate the results for DALO-t=1. We can see (a), (b), and (c) are better/more accurate in the p=1-optimal algorithm compared with DALO-t=1. Results (b) and (c) for the p=1-optimal algorithm become less accurate when the density increases. This is because the number of removed edges becomes large in the high density region. Figure 4(ii) shows the results for graphs with 20 nodes and density 0.3, varying the induced width. We can see even the induced width is

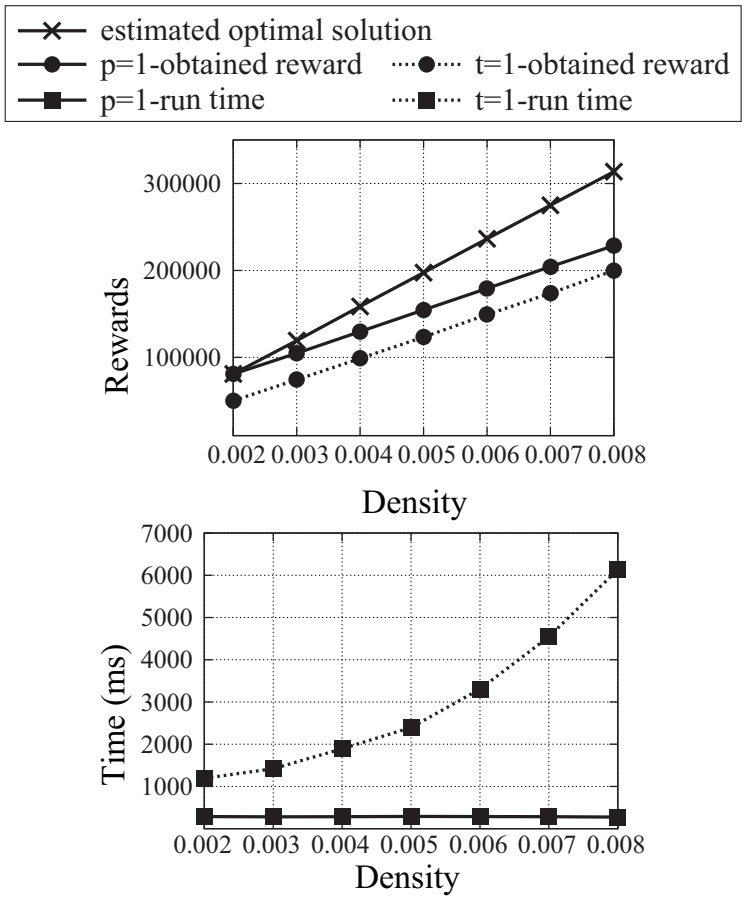


Fig. 5. Obtained rewards (not normalized) and run time (ms) for graphs with 1000 nodes and induced width 5. Broken line indicates results for DALO-t=1.

increased, (a), (b), and (c) for p=1-optimal algorithm are better/more accurate compared with DALO-t=1. Figure 4(iii) shows the results for graphs with density 0.3 and induced width 3, varying the number of nodes. The obtained results are similar to Fig. 4(i), i.e., (b) and (c) for the p=1-optimal algorithm become less accurate when the number of nodes increases.

Moreover, we show the results for large-scale problem instances. For them, obtaining an optimal solution is infeasible. Figure 5 shows the results for graphs with 1000 nodes and induced width 5, varying the density. Since we cannot obtain optimal solutions for these problem instances, we show the values of the obtained reward (which are not normalized). By setting the induced width to 5, we cannot create a graph whose density is greater than 0.01. We can see the

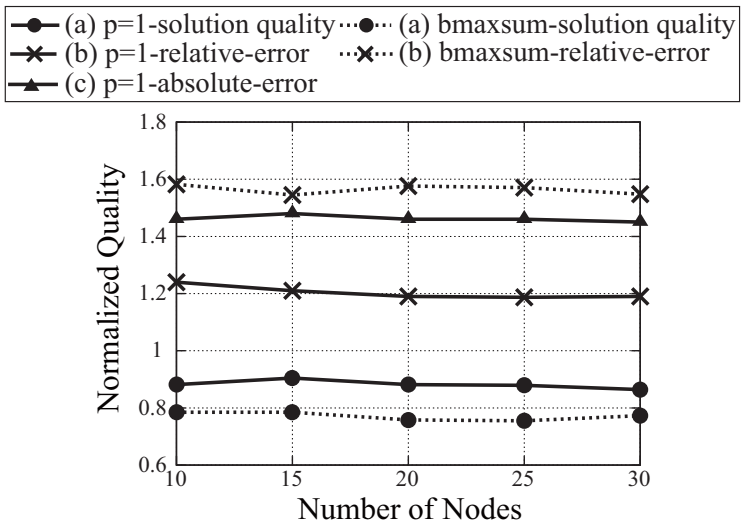


Fig. 6. (a), (b), and (c) in p=1-optimal algorithm and bounded max-sum algorithm for graphs with induced width 2. Broken line indicates results for bounded max-sum (bmaxsum) algorithm. Value closer to 1 is desirable.

rewards/run time for the p=1-optimal algorithm are greater/shorter compared to those for DALO-t=1.

Finally, we compare our algorithm for p=1-optimality and the bounded max-sum (bmaxsum) algorithm. We used graph coloring problems in the same settings presented in [9], except that the reward of each binary constraint is in the range [0,. . . ,6]. Figure 6 shows the results for graphs with induced width 2, varying the number of nodes. A value closer to 1 is desirable. Broken lines indicate the results for the bounded max-sum algorithm. We can see (a) and (b) (also (c)) are better in the p=1-optimal algorithm compared with the bounded max-sum algorithm.

One might imagine that the relative error of bounded-max-sum can be higher than the absolute error of p-optimality with p=1, since bounded max-sum algorithm uses some information about the real cost of removed edges. Since our algorithm and the bounded maxsum algorithm use different graph structures (i.e., a standard constraint graph and a factor graph, respectively), we cannot simply say that an informed/heuristic method for eliminating edges should work better. Our speculation is that the set of eliminated edges chosen by the bounded maxsum algorithm is somewhat far away from an optimal overall choice, since the elimination method is greedy/local.

We show the results for large-scale problem instances (graphs with 1000 nodes and induced width 2) in Fig. 7. Similar to the problem instances used in Fig. 5, obtaining an optimal solution is infeasible for these problem instances. By setting

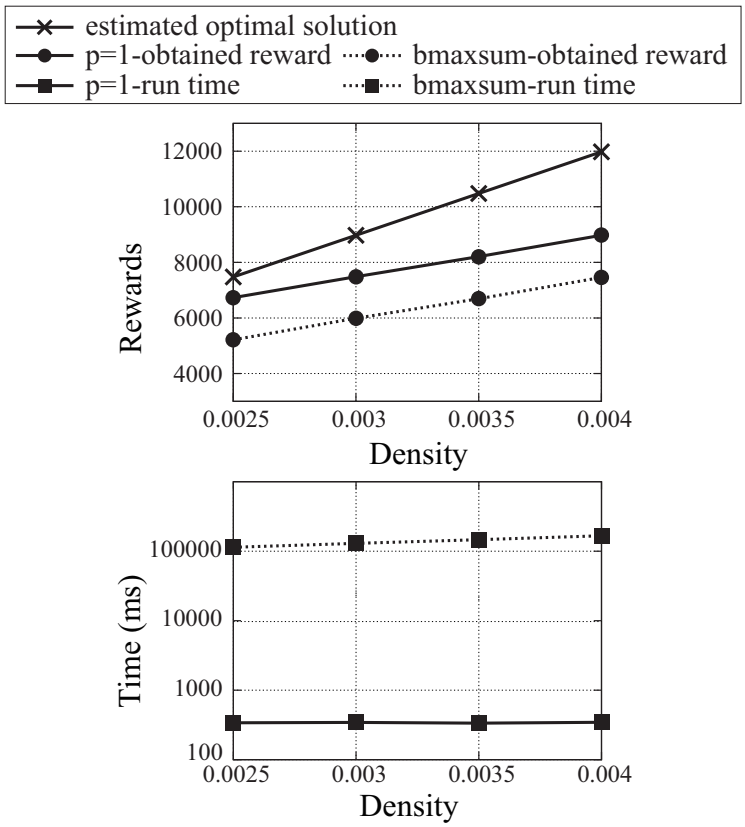


Fig. 7. Obtained rewards (not normalized) and run time (ms) for graphs with 1000 nodes and induced width 2. Broken line indicates results for bounded max-sum (bmax-sum) algorithm.

the induced width to 2, we cannot create a graph whose density is greater than 0.004. We show the values of the obtained reward (which are not normalized) as in Fig. 5. We can see the rewards/run time for the p=1-optimal algorithm are greater/shorter compared to those for the bounded max-sum algorithm.

In summary, these experimental results reveal that (i) the quality of the obtained solution of the p=1-optimal algorithm is much better compared with DALO-t=1 and bounded max-sum algorithms, (ii) the estimated quality of an optimal solution based on the absolute/relative error bounds for the p=1-optimal algorithm is more accurate than the other algorithms, and (iii) the run time of our algorithm is much shorter.

Although we did not show the results of ADPOP, they are basically similar to our algorithms, since these two algorithms differ only in the methods to de-

termine which edges to eliminate. The advantage of our algorithm is that it can provide the bound of a solution a priori.

Let us speculate why our algorithm can obtain better results compared to DALO-t and the bounded max-sum algorithm. These algorithms obtain approximate solutions of the original problem, while our algorithm obtains an *optimal* solution for a relaxed problem. If the relaxed problem is not so different from the original problem (e.g., the induced width is small), our algorithm can find a better solution quickly.

It must be mentioned that we require knowledge of the induced width and r_{max} to obtain a priori bound based on p-optimality. On the other hand, the error bound obtained by k/t-optimality is independent from problem instances. If r_{max} can be extremely large, while the average of the binary rewards is rather small compared to r_{max}, the absolute error bound of p-optimality becomes less informative.

5 Conclusion

We developed a new solution criterion called p-optimality and an incomplete algorithm for obtaining a p-optimal solution. This algorithm utilizes a graph structure called a pseudo-tree, which is widely used in complete DCOP algorithms. We provided the upper bounds of the absolute/relative errors of the solution, which can be obtained a priori/a posteriori, respectively. We showed that our algorithm for p=1-optimality can obtain better quality solutions and estimate more accurate error bounds compared with DALO-t for t=1-distance-optimality and the bounded max-sum algorithm. Furthermore, we showed that the run time for our algorithm for p=1-optimality is much shorter compared to these existing algorithms. Our future works include developing an anytime/complete algorithm that utilizes our algorithm as a preprocessing phase. A similar idea, i.e., using ADPOP as a preprocessing for ADOPT, is presented in [15].

References

[1] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[2] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[3] Roger Mailler and Victor Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems*, pages 446–453, 2004.

[4] Stephen Fitzpatrick and Lambert Meertens. Distributed coordination through anarchic optimization. In Victor Lesser, Charles Ortiz, and Milind Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer Academic Publishers, 2003.

[5] Jonathan Pearce, Milind Tambe, and Rajiv Maheswaran. Solving multiagent networks using distributed constraint optimization. In *AI Magazine, 29(3)*, pages 47–66, 2008.

[6] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars 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.

[7] Roie Zivan. Anytime local search for distributed constraint optimization. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 1449–1452, 2008.

[8] Christopher Kiekintveld, Zhengyu Yin, Atul Kumar, and Milind Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, 2010.

[9] Alessandro Farinelli, Alex Rogers, and Nicholas Jennings. Bounded approximate decentralised coordination using the max-sum algorithm. In *Proceedings of the 12th International Workshop on Distributed Constraint Reasoning*, pages 46–59, 2009.

[10] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *Proceedings of the Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pages 802–806, 2005.

[11] Jonathan Pearce and Milind Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1446–1451, 2007.

[12] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[13] Zhengyu Yin. USC dcop repository. University of Southern California, Department of Computer Science, 2008.

[14] Thomas Léauté, Brammert Ottens, and Radoslaw Szymanek. FRODO 2.0: An open-source framework for distributed constraint optimization. In *Proceedings of the 12th International Workshop on Distributed Constraint Reasoning*, pages 160–164, 2009.

[15] James Atlas, Matt Warner, and Keith Decker. A memory bounded hybrid approach to distributed constraint optimization. In *Proceedings of the 11th International Workshop on Distributed Constraint Reasoning*, pages 37–51, 2008.

Evolving Agent-Based Model Structures using Variable-Length Genomes

James Decraene, Mahinthan Chandramohan, Fanchao Zeng,
Malcolm Yoke Hean Low, and Wentong Cai

School of Computer Engineering,
Nanyang Technological University, Singapore 609479.
{jdecraene, chan0415, fczeng, yhlow, aswtcai}@ntu.edu.sg,
WWW home page: http://pdcc.ntu.edu.sg/EVOSIM/

Abstract. We present a novel evolutionary computation approach to optimize agent based models using a variable-length genome representation. This evolutionary optimization technique is applied to Computational Red Teaming (CRT). CRT is a vulnerability assessment tool which was originally proposed by the military operations research community to automatically uncover critical weaknesses of operational plans. Using this agent-based simulation approach, defence analysts may subsequently examine and resolve the identified loopholes. In CRT experiments, agent-based models of simplified military scenarios are repeatedly and automatically generated, varied and executed. To date, CRT studies have used fixed-length genome representation where only a fixed set of agent behavioural parameters was evolved. This may prevent the generation of potentially more optimized/interesting solutions. To address this issue, we introduce the hybrid variable-length crossover to evolve the structure of agent-based models. A maritime anchorage protection scenario is examined in which the number of waypoints composing the vessel's route is subjected to evolution. The experimental results demonstrate the effectiveness of our proposed method and suggest promising research avenues in complex agent-based model optimization.

Keywords: Agent-based simulation, multi-objective optimization, variable-length genome

1 Introduction

Computational Red Teaming is an agent-based simulation method which aims at identifying the critical weaknesses of military operational plans [16, 4]. A bottom-up/agent-based approach is thus adopted to analyse the complex dynamics that may emerge in combat systems. In CRT experiments, many agent-based model variants are executed/evaluated where two teams (a defensive "Blue" and belligerent "Red") are opposed using different tactical plans (as defined in the agent-based model specifications). The modelling and analysis of these tactical plans are automated and are conducted using evolutionary algorithms. The objectives of the evolutionary algorithms are, for instance, to generate Red tactical plans to best defeat Blue.

Through the analysis of optimized agent-based simulation models, one may identify Red tactical plans which may pose serious threats. Following on from this, defence analysts may attempt to resolve the operational weaknesses exposed through CRT. To our knowledge most CRT studies (see Section 2.2 for a brief survey) have focused on the examination of the agents' behaviour (e.g., aggressiveness, cohesiveness, determination, etc.). Such properties were subjected to evolutionary optimization. In these studies, the set of "evolvable model parameters" was fixed and commonly included less than 20 behavioural parameters.

We argue that such studies are limited when considering the optimization of *particular* military operations. Indeed, one may be interested in examining/generating complex courses of actions which cannot be encoded/evolved using a fixed set of behavioural parameter values. For instance, we may desire to optimize the agents' operational route where the number of waypoints may vary. Another example is the optimization of squad composition where both the number of agents and associated profile (e.g. engineer, infantry, sniper, etc) could be varied.

When considering the traditional fixed-length genome approach, one has to pre-determine and fix the number of, e.g., waypoints or agents. If this parameter is set too low, then this would prevent the emergence of optimal operational plans as the evaluated solutions are not "complex" enough. On the contrary, if such parameters are set too high, then this unnecessarily increases the complexity (through augmenting the search space dimensionality) of the search process and may prevent, as well, the finding of optimal solutions. Our proposed method attempts to deal with the optimization of such operational plans where a fixed-length genome approach may lead to optimality issues.

Novel techniques are required where additional simulation model properties, including the simulation model structure, are to be dynamically varied (i.e. added/removed) and evaluated. To extend and potentially enhance the CRT methodology, we investigate the evolution of agent-based model structures using variable-length genomes (through introducing a novel evolutionary computation technique coined the hybrid variable length crossover), in which additional simulation model properties (e.g., the model or distinct agent's structure) can be subjected to evolution.

To assist this research, we utilize a modular evolutionary framework coined CASE (complex adaptive systems evolver). Multi-objective evolutionary computation techniques are utilized to optimize the agent-based models.

Background material on agent-based models for military applications and CRT are first presented. A survey on variable length genome techniques for evolutionary algorithms follows. The CASE framework is then detailed. Experiments, using CASE and the agent-based platform MANA, are then conducted to evaluate the application of variable length genomes for the evolution/optimization of agent-based model structures. The experiments consider a simplified CRT maritime anchorage protection scenario. Finally, we conclude the paper and outline future research directions which may merit investigations to develop this work. This paper is a direct follow-up of the preliminary study reported in [8] where

agent-based model structures were evolved using a *fixed*-length genome representation.

2 Background

We first briefly describe some agent-based simulations that have been applied to military operations research. Then the Computational Red Teaming concept is presented.

2.1 Military Agent-Based Simulations

Agent Based Simulations have recently attracted significant attention to model the intricate and non-linear dynamics of warfare. Combat is thus here conceptually regarded as a complex adaptive system which components (i.e. the battlefield, soldiers, vehicles, etc.) are modelled using a bottom-up agent-based approach. The agents' computational methods may include stochastic processes resulting in a stochastic behaviour at the system level. Examples of ABS applied to Military Decision Making include: ISAAC/EINSTein [16], CROCADILE [12], WISDOM [26], MANA [18] and Pythagoras [1]. A review of ABS applied to various military applications is provided by Cioppa et al [6].

These systems have been specifically devised to simulate defence related scenarios in which the properties of the environment and the Red/Blue teams may be specified [20]. The level of representation/abstraction (e.g., number of spatial dimensions, range of agents' properties, type of vehicles, etc.) varies among these ABS systems. Although the level of accuracy in representing real world environments/individuals may not faithfully reflect reality, it is argued that such ABS models account for the key features (e.g., local interactions between agents) necessary to exhibit complex emerging phenomena/behaviour at the system level which are typical of real battlefields [16]. Thus, these "distillation" models can expose the emerging phenomena of interest without the burden of modelling and simulating unnecessary complex features (e.g., gravity, wind, detailed physics of distinct simulated agents/weapons/vehicles, etc.). Agent-based modelling is one of the key technologies supporting Computational Red Teaming which is described in the next section.

2.2 Computational Red Teaming

Computational Red Teaming (CRT) combines agent-based simulations and evolutionary computation (EC) techniques as follows. CRT exploits EC techniques to evolve simulation models to exhibit pre-specified/desirable output behaviors (i.e., when Red defeats Blue). To date, most CRT studies have only addressed the evolution of a fixed set of agent parameters (e.g., troop clustering/cohesion, response to injured teammates, aggressiveness, stealthiness, etc.), defining the behaviour or personality of the Red team. These parameters are evolved to optimize the Red agents collective efficiency (e.g., maximize damage to target

facilities) against the Blue team. Example studies include: [16, 26, 4, 19]. These studies demonstrated the promising potential of CRT systems to automatically identify the Blue team's weaknesses.

Further CRT investigations adopted a co-evolutionary approach where the set of behavioural parameter values of both teams are coevolved. This arms race approach complements the previous one by automating the analysis required to improve the Blue team's defence operational plan against the adaptive Red team. Examples of co-evolutionary CRT studies can be found in [17, 22, 5]. This approach enables one to generate operational tactics that are more efficient and robust against a larger range of scenarios. Nevertheless a trade-off exists in terms of robustness over efficiency according to the range of confronted Red tactics (i.e., the evolved tactics only yield average performances against multiple Red tactics).

The extension of one-sided to co-evolutionary CRT significantly increases the search spaces allowing for the exploration of more diverse simulation models. As the diversity of evaluated simulation models is increased, a wider range of potentially critical scenarios may be identified. Exploring more diverse scenarios enables one to devise more robust and effective defensive strategies against potential threats and adaptive adversaries. Nevertheless, the expansion of this search space is associated with a *dramatic* increase in computational cost. Also, due to this high effect on computational complexity, no Pareto-based multi-objective co-evolutionary approaches to CRT have been proposed to date. In this paper, we focus on the multi-objective structural evolution of agent-based models where only Red is evolved against Blue.

Finally, none of the above studies has attempted to evolve agent-based model structures. We here propose a novel method to dynamically vary the range of evolvable parameters through varying the candidate solutions' genome string length. In the next section, we survey some related evolutionary computation approaches which focused on variable-length genome techniques.

3 Survey of Variable Length Genome Techniques

Several studies have investigated variable-length genomes in the context of genetic algorithms. None of these schemes has been applied, to the authors' knowledge, to vary the structure of genomes which encode for agent-based model specifications.

3.1 Messy Genetic Algorithm

An early attempt addressing variable-length genomes was proposed by the Messy Genetic Algorithm (m-GA) [13]. In m-GA, the classical one-point crossover operator is replaced by the "cut" and "splice" operators. The cut operator is first applied upon each parent genome string where a locus point is selected at random on each genome, cutting each string into two strings. The splice operator is employed to rejoin the resulting four strings in a random order. The cut and

splice operators were applied upon bit strings and is thus not directly applicable to the real-valued genomes used in Computational Red Teaming experiments.

This seminal work inspired the crossover techniques for variable-length genomes presented in the next sections.

3.2 The Speciation Adaptation Genetic Algorithm

The Speciation Adaptation Genetic Algorithm (SAGA) was introduced by Harvey [14]. In m-GA, strings were recombined regardless of the strings' contents. In contrast, SAGA was proposed to maximize the similarity between strings that are recombined to diminish undesirable disruptive effects that may occur when using a "blind" cut and splice method.

The similarity of the two parent genome strings is computed using the Longest Common Subsequence (LCSS) metric. The LCSS is the longest uninterrupted matching substring of gene values (alleles) found between two strings of arbitrary length. In SAGA, a random crossover point is chosen on the first parent string, then the algorithm tests every possible crossover point on the second string. For each potential crossover point, the algorithm calculates the LCSS sum on both the left and right regions of the genomes. The second parent string is cut at the crossover point with the highest LCSS score. If multiple crossover points are eligible, then one is selected at random.

3.3 Virtual Virus

Similarly to SAGA, the Virtual Virus (VIV) crossover [2] is based on the similarity between parent genome strings. In contrast with SAGA, VIV can only be applied upon similar sequences.

In VIV, the probability of crossover is governed by the level of similarity between the parent genome strings. This level of similarity is determined by the number of matched alleles between parent strings within a pre-specified fixed size window. As in SAGA, a random crossover locus point is selected on one of the parent strings. VIV then compares the sequence of alleles (limited by the window size) from this selected point with all possible substrings of the same size on the other parent string. The substring position that includes the greatest number of matched alleles is then recorded. The strings are then cut within the matched substring given a similarity-based probability.

3.4 Synapsing Variable-Length Crossover

In both SAGA and VIV crossover operators, the crossover locus point was first selected in *one* of the parent strings, then a relatively similar substring was searched for in the second parent string. Thus the first selected string was utilized as a template. In contrast, the Synapsing Variable-Length Crossover (SVLC) [15] employs both parent strings as a template. The motivation is to preserve any common sequences between the parent strings, where only differences are to be exchanged during recombination.

In SVLC, the level of similarity between parent strings is computed using a variant version of the LCSS (used in SAGA). A major difference with the previous crossover techniques is that SVLC also includes mutation operators (which are individually applied on children genome strings) which may affect the genome length. These length varying mutation operators were implemented in addition to the traditional point mutation operators. Four length varying mutation operators are distinguished as follows: 1) A random sequence of alleles is inserted at a random locus point on the genome string, 2) a genome substring is selected/removed at random, 3) a substring is selected at random and duplicated at a random locus point and 4) a substring is selected at random and duplicated at the beginning or end of the genome string. Various probabilities were pre-defined for each mutation operator (most disruptive operators, such as the substring insertion, were assigned a significantly lower probability of occurring).

3.5 NeuroEvolution of Augmenting Topologies

In NeuroEvolution of Augmenting Topologies (NEAT) [23], the structural evolution of artificial neural networks was investigated. NEAT included an evolutionary scheme which accounted for a variable-length genome representation. The key idea of NEAT is evolutionary complexification where (initially simple) structures/genome strings would *incrementally* complexify (as determined by the number of network nodes/interactions) through evolution.

A benefit of NEAT is to minimize the dimensionality (number of genes) through complexification. Indeed, the evolutionary process would evaluate genome strings of higher dimensionality only if these structures yield higher fitness values. This enables NEAT to search through a minimal number of genes, significantly reducing the number of generations necessary to find competitive solutions, and ensuring that genome strings are not more complex than necessary.

A historical marking technique was implemented to identify the similarities between genome strings. This marking was also used to perform the recombinations. The mutation operators included a gene duplication method (a similar length varying mutation operator was implemented in SAGA). In contrast with the bitstring representation of genomes in the previous approaches, NEAT relies on a real-valued genome representation. Alterations of the gene values were conducted using the mutation operators and not through recombinations.

3.6 Summary

This section summarises the above techniques and attempts to identify the most promising computational techniques. The latter will be then considered and evaluated in our study on the evolution of agent-based models using variables length genomes.

m-GA uses a simple cut and splice implementation which ignored any similarities between parent genome strings. SAGA and VIV accounted for similarities between the parent genome strings, however, recombinations were heavily based

on the first selected parent genome string (i.e. the template) where no appropriate crossover points could be found in the second parent string. This may result in disruptive outcomes (i.e. loss of information). SVLC resolved these issues through considering both parent genome strings as templates. Moreover, SVLC introduced length-varying mutation operators. NEAT is, to some extent, similar to SVLC but took an evolutionary complexification approach where genome strings progressively increase in complexity/length through evolution.

When evolving agent-based models, the model specifications are encoded as real-valued genome strings (where each value encodes for a specific agent behavioural parameter). This conflicts with the bitstring encoding representation of m-GA/VIV/SAGA and SVLC. NEAT used a real-valued genome representation but the crossover operator does not directly modify the gene values through recombinations. In real-valued fixed-length genome evolutionary algorithms, the Simulated Binary Crossover [7] has long been established as an efficient method to recombine such genome strings encoded in continuous space. SBX will be considered into the novel hybrid method proposed in Section 5.

Moreover, the varying length mutation operators proposed in SVLC and NEAT presented promising outcomes to dynamically evolve the structure of genome strings. Such operators will be examined in our hybrid crossover method. Finally the evolutionary complexification approach of NEAT may yield potential benefits as it would avoid the exploration/evaluation of unnecessarily complex genome strings (i.e. reducing computational efforts). Nevertheless evolutionary complexification is not explored here but will be investigated in future work.

4 The Evolutionary Framework

A detailed description of the evolutionary framework, coined CASE (complex adaptive systems evolver), is provided in this section. This framework was also described and evaluated (against additional system features such as optimization under constraint, multi-objective optimization and cloud computing) in [11, 10, 9].

The CASE framework was inspired by the Automated Red Teaming framework [4] which was developed by the DSO National Laboratories of Singapore. In contrast with DSO's system (which was dedicated to examining military simulation models), we aim at providing a relatively more flexible and platform-independent system capable of evolving simulation models for a wider variety of application domains.

CASE is composed of three main components which are distinguished as follows:

1. *The model generator*: This component takes as inputs a base simulation model specified in the eXtended Markup Language and a set of model specification text files. According to these inputs, new XML simulation models are generated and sent to the simulation engine for evaluation. Thus, as currently devised, only simulation models specified in XML are supported. Moreover, the model generator may consider constraints over the evolvable

parameters (this feature is *optional*). These constraints are specified in a text file by the user. These constraints (due for instance to interactions between evolvable simulation parameters) aim at increasing the plausibility of generated simulation models (e.g., through introducing cost trade-off for specific parameter values).

2. *The simulation engine*: The set of XML simulation models is received and executed by the stochastic simulation engine. Each simulation model is replicated a number of times to account for statistical fluctuations. A set of result files detailing the outcomes of the simulations (in the form of numerical values for instance) are generated. These measurements are used to evaluate the generated models, i.e., these figures are the fitness (or "cost") values utilized by the evolutionary algorithm (EA) to direct the search.
3. *The evolutionary algorithm*: The set of simulation results and associated model specification files are received by the evolutionary algorithm, which in turns, processes the results and produce a new "generation" of model specification files. The generation of these new model specifications is driven by the user-specified (multi)objectives (e.g., maximize/minimize some quantitative values capturing the target system behaviour). The algorithm iteratively generates models which would incrementally, through the evolutionary search, best exhibit the desired outcome behaviour. The model specification files are sent back to the model generator; this completes the search iteration. This component is the key module responsible for the automated analysis and modelling of simulations.

The above components are depicted in Figure 1 which presents the flowchart of an example experiment. Further details about the input files settings can be found in [9]. Finally, a demonstration video of CASE can be visualized at `http://www.youtube.com/watch?v=d2Day_MEruc`.

5 Hybrid Variable Length Crossover

As discussed earlier (Section 3.6), we incorporate a number of existing evolutionary computation techniques to implement our crossover technique for variable-length genomes. We propose the hybrid variable length crossover (HVLC), which is a combination of both SBX and one point crossover (where similarities between parent genome strings are considered).

In HVLC, two distinct regions within a genome string (Fig. 2) are distinguished: A static sequence of genes (which is located at the beginning of the genome string) and dynamic sequence of genes which may vary in length.

The SBX crossover [7] was designed to recombine fixed-length genomes, thus it cannot be directly used for variable-length genomes. SBX is here utilized to recombine common substrings (which includes the static genome string region and any other sequences, with equal sizes, of "matched" genes encoding for identical model properties).

During the crossover operation, a crossover point is randomly selected (at a valid locus point, so that no structures are broken, see Fig. 3) upon the common

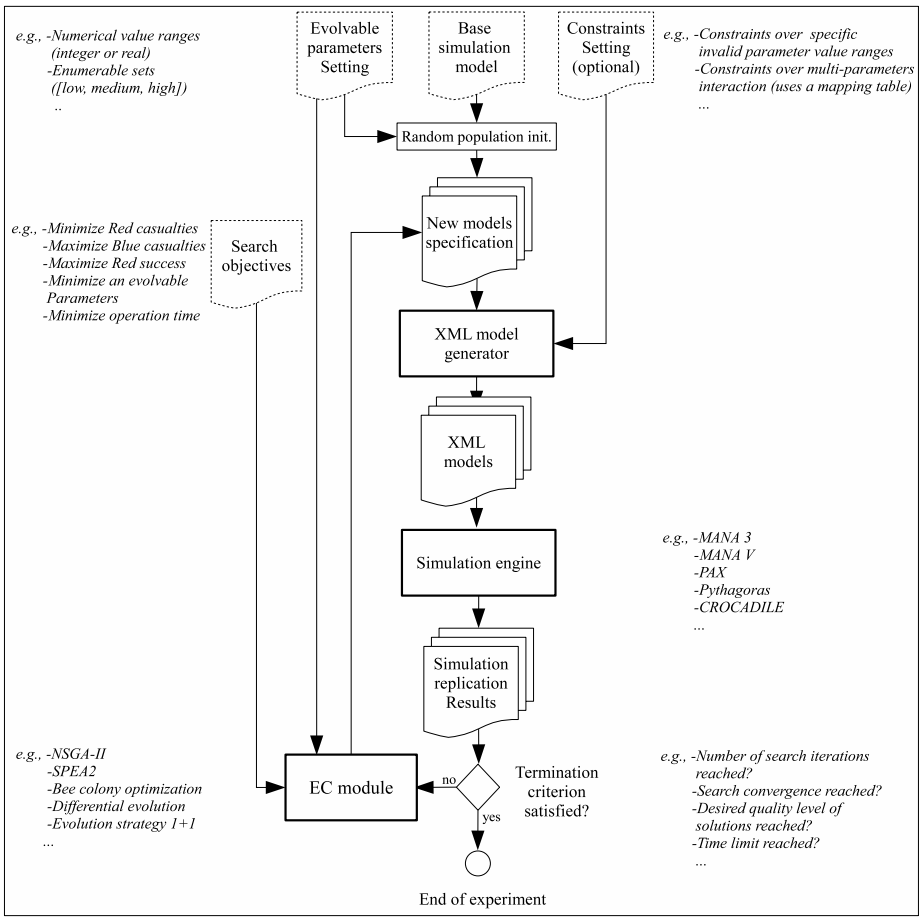


Fig. 1. Flowchart of an example experiment. The dashed documents distinguish the user inputs. Using the base XML model, a population of randomly generated model variants is first created. The initial parameter values are randomly generated using a uniform distribution and are bounded by the evolvable parameters setting file provided by the user. Both the simulation engine and evolutionary computation module call external libraries and/or binaries. The XML model generator employs the Libxml library (`http://libxml.rubyforge.org`) to parse and generate XML models. The constraint setting file is utilized by the XML model generator to apply user-defined constraints over the evolvable parameters.

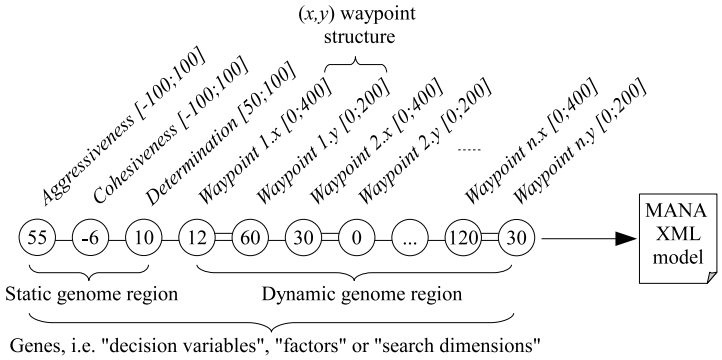


Fig. 2. Variable Genome String Representation. This genome string illustrates the encoding of simulation models utilized in the experiments reported in Section 6. In this example the waypoint structures (composed of a pair of genes encoding for spatial coordinates) are dynamically varied (removed/added) during the evolutionary search, reducing/expanding the length of the genome string. Double-linked genes indicate structures that cannot be broken through recombination.

genetic sequences of both parent genome strings. The genome strings are then cut and spliced as in m-GA. Then the SBX operator is applied over the common sequences of genes.

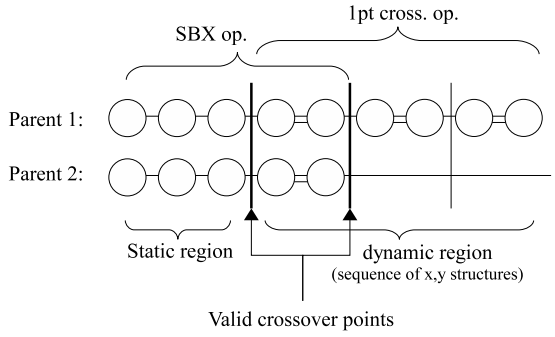


Fig. 3. Example HVLC crossover operation. Valid crossover points are distinguished to disable the recombination of genetic sequences encoding structures. In this example, waypoints are such "unbreakable" structures composed of two distinct genes encoding for x, y spatial coordinates.

To vary the genome length, we propose a length varying mutation operator in which two types of mutation can be distinguished (Fig. 4):

1. *Deletion*: A gene (or structure composed of multiple genes) is removed from the end of the genome string (reducing the genome string length).

2. *Duplication*: A gene (or structure composed of multiple genes) is selected and duplicated at the end of the genome string (increasing the genome string length).

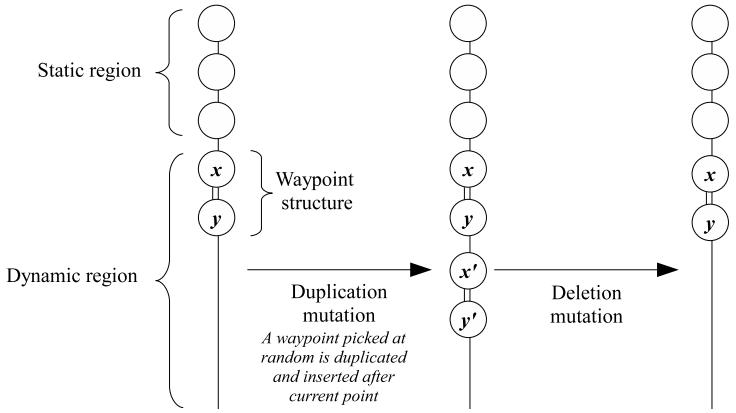


Fig. 4. Example HVLC mutation operations. In this example, the duplication and deletion mutations are applied upon waypoint structures. As a result, the entire genetic sequence representing these structures are dynamically duplicated/deleted within the genome string.

The probability of each of these operators being applied to any children genome string is 0.01. Finally, the polynomial mutation operator is also applied to each gene, introducing further variations upon the gene values.

6 Experiments

We report a series of experiments using the CASE framework and the agent-based simulation platform MANA [18]. A single case study is here examined in which the agents' structure or more specifically the number of waypoints and associated coordinates determining the agents' routes are subjected to the evolutionary process. Although examining a unique case study *considerably* limits the significance of the experimental results (the authors acknowledge that further case studies must be examined for a better appreciation of the results), we limit the current investigation to a single case study as this particular scenario was previously studied in multiple publications [24, 19, 25, 8]. This paper extends the work that has been conducted in this well-studied model scenario. Future work will include other case studies to complement our investigation and understanding on variable-length genomes for the structural evolution of simulation models.

6.1 The model

The maritime anchorage protection scenario was originally proposed by a team of defence analysts and academic researchers [24] and further developed in [19, 25, 8]. In this scenario, a Blue team (composed of 7 vessels) conducts patrols to protect an anchorage (in which 20 Green commercial vessels are anchored) against threats. Single Red vessel attempts to break Blues defence tactics and inflict damages to anchored vessels. The aim of the study is to discover Reds strategies that are able to breach through Blues defensive tactic. Fig. 5 depicts the scenario which was modelled using the ABS platform MANA.

In [8], a preliminary study on "evolvable simulation" (i.e. where the structure of the model is evolved) was examined. In this work, a fixed-length genome was employed, additional genes were introduced to control the number of waypoints to be "switched on". Thus, the maximum number of waypoints that compose the Red vessel route had to be pre-specified (this determines the genome string length). The current study extends this preliminary work through removing such "control" genes and effectively vary dynamically the genome string length.

6.2 Experimental Setting

In CASE, each candidate solution (a distinct simulation model) is represented by a vector of real values defining the different evolvable Red behavioural parameters (Table 1). As the number of decision variables increases, the search space becomes dramatically larger.

The selection scheme (based on the crowding distance and Pareto sorting) of the Non-dominated Sorting Algorithm II (NSGAII) is employed to assist the evolutionary search. This algorithm is executed using the following parameters: population size = 100, number of search iterations = 200, mutation probability = 0.1, mutation index = 20, crossover rate = 0.9 and crossover index = 20. Such parameter values for NSGAII are commonly used in the literature to solve two-objective optimization problems. The population size and number of search iterations indicate that 20,000 distinct MANA simulation models are generated and evaluated for each experimental run. Each individual simulation model is executed/replicated 30 times to account for statistical fluctuations (30 replications would approximately take 10 wallclock seconds to execute using an Intel Dual Core CPU @ 2.66GHZ).

The efficiency of the search is measured by the number of Green casualties with respect to the number of Red casualties. In other words, the search objectives are:

- To minimize the number of Green (commercial) vessels "alive".
- To minimize the number of Red casualties.

Considering the current scenario, these objectives are thus conflicting. Moreover, the true Pareto front is here unknown. In the next section we report the experimental results using the above model.

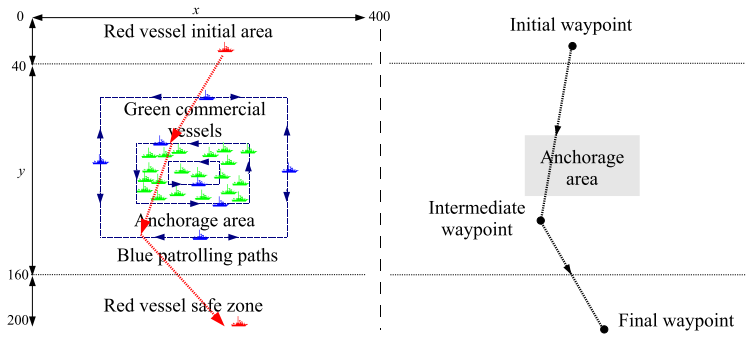


Fig. 5. Schematic overview of the case study . The map covers an area of 100 by 50 nautical miles (1 nm = 1.852km). 7 Blue vessels conduct patrols to protect an anchorage of 20 Green commercial vessels against a single Red vessel. The Red vessel intends to break the Blues defence, inflict damages to the anchored Green vessels and finally, to escape to the Red vessel safety area. **Left**: The dashed lines depict the patrolling paths of the different Blue vessels. The Blue patrolling strategy is composed of two layers: an outer (with respect to the anchorage area, 30 by 10 nm) and inner patrol. The outer patrol consists of four smaller but faster boats. They provide the first layer of defence whereas the larger and heavily armoured ships inside the anchorage are the second defensive layer. The Red craft was set up to initiate its attack from the north. The initial positions of Blue vessels are fixed. In contrast, the Green commercial vessels' initial positions are randomly generated within the anchorage area at each MANA execution. **Right**: Example Red route. Home waypoint (Home WP) is constrained to the distinct agent's initial area. Similarly, the final waypoint is to be located in the opposite area. Intermediate waypoints occur in the remaining middle area. Note that in the below experiments, we dynamically evolve the number of intermediate waypoints. Whereas the coordinates of all waypoints, including the home and final ones, are subjected to evolution.

7 Results

To evaluate the quality of the (multi-objective) solutions through the evolutionary search, the hypervolume indicator [27] is utilized. This method is currently considered as the state of the art technique to evaluate Pareto fronts. This indicator measures the size of the objective space subset dominated by the Pareto front approximation.

In Fig. 6, the HVLC is compared with the fixed-length genome approach studied in [8] using NSGAII. Whereas the numerical values resulting from the evolutionary experiments are shown in Table 2.

In Fig. 6 and Table 2, it can be observed that HVLC consistently outperformed its fixed-length genome counterpart. When comparing the best Pareto set approximations (Fig. 7) resulting from both sets of evolutionary runs, comparable results were achieved. HVLC was nevertheless more consistent throughout the 10 distinct evolutionary runs (when considering the mean hypervolume indicator value and standard deviation) in achieving competitive results.

(a) Fixed Blue parameters

Parameter	Value
Detection range (nm)	24
# hits to be killed	2
Weapon hit prob.	0.8
# patrolling agents	7
Speed (unit)	100
Weapon range (nm)	8
Determination	50∨0
Aggressiveness	0∨100
Cohesiveness	0

(b) Fixed Red parameters

Parameter	Value
Detection range (nm)	8
# hits to be killed	1
Weapon hit prob.	0.8
# agents	5
Speed (unit)	100
Weapon range (nm)	5

(c) Evolvable Red paramaters

Parameter	Min	Max
Vessel home position(x,y)	(0,0)	(399,39)
Intermediate waypoint position (x,y)	(0,40)	(399,159)
Vessel final position (x,y)	(0,160)	(399,199)
Determination	20	100
Aggressiveness	-100	100
Cohesiveness	-100	100

Table 1. (a): Fixed Blue parameters. Value pairs are specified for the determination and aggressiveness properties. In this model, Blue changes its behaviour upon detecting Red, i.e., Blue "targets" Red, with aggressiveness being increased, when the latter is within Blue's detection range. (b): Fixed Red parameters. The behavioural parameters are not specified as these parameters are subjected to evolution. (c): Evolvable Red parameters: As mentioned earlier, the intermediate waypoint structures are dynamically inserted/removed within the agent-based model during the evolutionary search. The final positions of the Red craft is constrained to the opposite region (with respect to initial area) to simulate escapes from the anchorage following successful attacks. Behavioural or "psychological" elements are included in the evolvable decision variables. The aggressiveness determines the reaction of individual vessels upon detecting an adversary. Cohesiveness influences the propensity of vessels to maneuver as a group or not, whereas determination stands for the agent's willingness to follow the defined routes (go to next waypoint). The Red vessels' aggressiveness against the Blue patrolling force are varied from unaggressive (-100) to very aggressive (100). Likewise, the cohesiveness of the Red crafts are varied from independent (-100) to very cohesive (100). Finally, a minimum value of 20 is set for determination to prevent inaction from occurring.

Although the above preliminary results suggest a somewhat promising potential for the variable-length genome approach, only a single case study was here considered. As mentioned in the introduction, we do expect that this method may only benefit scenarios in which the structural evolution of simulation models is relevant (i.e. where the evolutionary experiment is not *constrained* to a set of evolvable parameters). Our future work will include a broader set of scenario

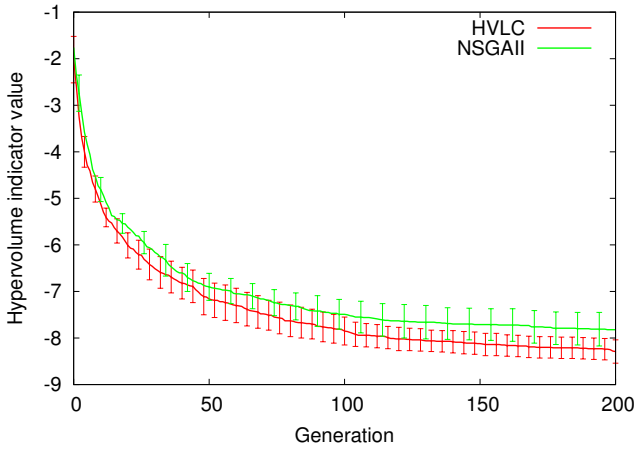


Fig. 6. Hypervolume volume dynamics. The lines identify the hypervolume indicator value averaged over 10 individual evolutionary runs using unique seeds. The error bars stands for the confidence interval (with $\alpha = 0.05$). The negative value of the hypervolume indicator is utilized for consistency with the *cost minimization* approach used in the experiments.

Table 2. Pareto optimality performance

Algo.	Best	Mean
NSGAII	-8.9249	-7.8211 $\pm$ 0.36
HVLC	-8.9995	**-8.2854 $\pm$ 0.25**

The bold values identify the best overall Pareto optimal approximation sets (when considering both the mean and a 95% confidence interval).

to better evaluate HVLC against existing evolutionary computation techniques such as the NSGAII.

In the remainder of this section, we discuss a potential explanation for the dynamics observed in the experiments using the fixed-length genome approach: A potential drawback of the fixed-length genome representation is the *epistasis* phenomenon. In biology, epistasis refers to non-linear interactions occurring between genes. It is currently hypothesized that epistasis may emphasize the "ruggedness" of the fitness landscape [3], leading to an increased level of difficulty for the evolutionary search. Note that epistasis may already occur *implicitly* between genes (here simulation model parameters) according to their specific values. The specification of control genes *explicitly* introduce epistatic interactions, which may harden the search difficulty level.

Indeed, a slight mutation in the value of "control" or epistatic genes, using the fixed-length genome approach, would result in large phenotype changes, where many waypoints may be turned off or on simultaneously. This clearly introduces non-linearities in the evolutionary search process. The level of epistatic

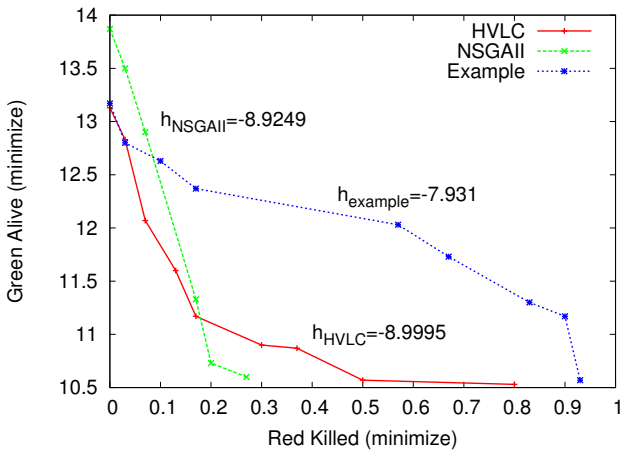


Fig. 7. Best Pareto set approximations resulting from the 10 distinct evolutionary runs conducted using NSGAII and HVLC. A third example Pareto front is shown to illustrate how a difference of 1 in the hypervolume indicator value may affect the Pareto front approximation quality.

interactions would moreover increase according to the pre-specified maximum number of waypoints. This ultimately limits the utilization of the fixed-length genome representation for such agent-based model optimization applications.

The results suggest that the variable-length genome representation may potentially alleviate this epistatic issue, leading to better Pareto optimality performances. Although these results are promising, further experiments remain required to investigate the potential benefit of the variable-length genome representation approach. Future work include the evolution of simulation models in which a large number of model structures is evolved. We hypothesize that the fixed-length genome approach would rapidly and significantly be outperformed by HVLC when tackling larger search problems including a relatively high level of explicit epistatic interaction. Methods which may quantify epistasis (as used in molecular biology research [21]) would also assist this future research.

Also the evolutionary complexification concept proposed by Stanley and Miikkulainen [23] will be investigated as it may reduce the number of search generations (i.e. optimizing the search convergence speed) through avoiding the evaluation of unnecessary complex simulation models.

8 Conclusions

The Computational Red Teaming methodology and related supporting technologies were first introduced. A survey on variable-genome length techniques for evolutionary computation was then presented. The evolutionary framework CASE was briefly described and utilized using a novel variable-length compu-

tational technique coined the hybrid variable length crossover. A series of experiments was conducted in which the structure of a simplified military agent-based model was evolved. HVLC was compared against a fixed-length genome approach. The experimental results suggested that our variable-length genome approach is a promising technique which, in overall, achieved better Pareto optimality performances than using fixed-length genomes. Nevertheless, this potential benefit must be further examined in future work where supplementary evolutionary experiments of differing complexity will be conducted.

Acknowledgements

We would like to thank the Defence Research and Technology Office, Ministry of Defence, Singapore, for sponsoring the *Evolutionary Computing Based Methodologies for Modeling, Simulation and Analysis* project which is part of the Defence Innovative Research Programme FY08.

References

1. Bitinas, E.J., Henscheid, Z.A., Truong, L.V.: Pythagoras: A New Agent-based Simulation System. Technology Review pp. 45–58 (2003)
2. Burke, D., De Jong, K., Grefenstette, J., Ramsey, C., Wu, A.: Putting More Genetics Into Genetic Algorithms. Evolutionary Computation 6(4), 387–410 (1998)
3. Choi, S., Jung, K., Moon, B.: Lower and Upper Bounds for Linkage Discovery. Evolutionary Computation, IEEE Transactions on 13(2009), 201–216 (2009)
4. Choo, C.S., Chua, C.L., Tay, S.H.V.: Automated Red Teaming: a Proposed Framework for Military Application. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. pp. 1936–1942. ACM (2007)
5. Choo, C.S., Chua, C.L., Low, K.M.S., Ong, W.S.D: A Co-evolutionary Approach for Military Operational Analysis. In: GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation. pp. 67–74. ACM (2009)
6. Cioppa, T.M., Lucas, T.W., Sanchez, S.M.: Military Applications of Agent-based Simulations. In: Proceedings of the 36th Winter Simulation Conference. pp. 171–180. ACM (2004)
7. Deb, K., Agrawal, R.: Simulated Binary Crossover for Continuous Search Space. Complex Systems 9(2), 115–148 (1995)
8. Decraene, J., Chandramohan, M., Low, M.Y.H., Choo, C.S.: Evolvable Simulations Applied to Automated Red Teaming: A Preliminary Study. In: Proceedings of the 42th Winter Simulation Conference. pp. 1444–1455. ACM (2010)
9. Decraene, J., Low, M.Y.H., Zeng, F., Zhou, S., Cai, W.: Automated Modeling and Analysis of Agent-based Simulations using the CASE Framework. In: In Proceedings of 11th International Conference on Control, Automation, Robotics and Vision (ICARCV). pp. 346–351. IEEE (2010)
10. Decraene, J., Yong, Y.C., Low, M.Y.H., Zhou, S., Cai, W., Choo, C.S.: Evolving Agent-based Simulations in the Clouds. In: Third International Workshop on Advanced Computational Intelligence (IWACI). pp. 244–249. IEEE (2010)

11. Decraene, J., Zeng, F., Low, M.Y.H., Zhou, S., Cai, W.: Research Advances in Automated Red Teaming. In: Proceedings of the 2010 Spring Simulation Multiconference (SpringSim). pp. 47:1–47:8. ACM (2010)
12. Easton, A., Barlow, M.: CROCADILE: An Agent-based Distillation System Incorporating Aspects of Constructive Simulation. In: Proceedings of the SimTecT 2002 Conference. pp. 233–238 (2002)
13. Goldberg, D., Korb, B., Deb, K., et al.: Messy Genetic Algorithms: Motivation, Analysis, and First Results. Complex systems 3(5), 493–530 (1989)
14. Harvey, I.: The SAGA Cross: The Mechanics of Recombination for Species with Variable Length Genotypes. In: Männer, R., Manderick, B. (eds.) In Proceeding of the Parallel Problem Solving from Nature 2. pp. 269–278. Elsevier (1992)
15. Hutt, B., Warwick, K.: Synapsing Variable-length Crossover: Meaningful Crossover for Variable-length Genomes. IEEE transactions on evolutionary computation 11(1), 118–131 (2007)
16. Ilachinski, A.: Artificial war: Multiagent-based Simulation of Combat. World Scientific Pub Co Inc (2004)
17. Kewley, R.H., Embrechts, M.J.: Computational Military Tactical Planning System. IEEE Transactions on Systems, Man, and Cybernetics, Part C 32(2), 161–171 (2002)
18. Lauren, M., Stephen, R.: Map-aware Non-uniform Automata (MANA)-A New Zealand Approach to Scenario Modelling. Journal of Battlefield Technology 5, 27–31 (2002)
19. Low, M.Y.H., Chandramohan, M., Choo, C.S.: Multi-Objective Bee Colony Optimization Algorithm to Automated Red Teaming. In: Proceedings of the 41th Winter Simulation Conference. pp. 1798–1808. ACM (2009)
20. Lucas, T.W., Sanchez, S.M., Martinez, F., Sickinger, L.R., Roginski, J.W.: Defense and Homeland Security Applications of Multi-agent Simulations. In: Proceedings of the 39th Winter Simulation Conference. pp. 138–149. IEEE (2007)
21. Matsuura, T., Kazuta, Y., Aita, T., Adachi, J., Yomo, T.: Quantifying Epistatic Interactions among the Components Constituting the Protein Translation System. Molecular Systems Biology 5(297) (2009)
22. McDonald, M.L., Upton, S.C.: Investigating the Dynamics of Competition: Coevolving Red and Blue Simulation Parameters. In: Proceedings of the 37th Winter Simulation Conference. pp. 1008–1012. ACM (2005)
23. Stanley, K., Miikkulainen, R.: Competitive Coevolution through Evolutionary Complexification. Journal of Artificial Intelligence Research 21(1), 63–100 (2004)
24. Wong, A.C.H., Chua, C.L., Lim, Y.K., Kang, S.C., Teo, C.L.J., Lampe, T., Hingston, P., Abbott, B.: Team 1: Applying Automated Red Teaming in a Maritime Scenario. In: In Scythe 3: Proceedings and Bulletin of the International Data Farming Community. pp. 3–5 (2007)
25. Xu, Y.L., Low, M.Y.H., Choo, C.S.: Enhancing Automated Red Teaming with Evolvable Simulation. In: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation. pp. 687–694. ACM (2009)
26. Yang, A., Abbass, H., Sarker, R.: Characterizing Warfare in Red Teaming. IEEE Transactions on Systems, Man, and Cybernetics, Part B 36(2), 268–285 (2006)
27. Zitzler, E., Brockhoff, D., Thiele, L.: The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In: Proceedings of The 4th International Conference on Evolutionary Multi-criterion Optimization, Lecture notes in computer science. vol. 4403, pp. 862–876. Springer (2007)

Reward-based region optimal quality guarantees

Meritxell Vinyals[1], Eric Shieh[2], Jesus Cerquides[1], Juan Antonio Rodriguez-Aguilar[1], Zhengyu Yin[2], Milind Tambe[2], and Emma Bowring[3]

[1] Artificial Intelligence Research Institute (IIIA),
Campus UAB, Bellaterra, Spain
{meritxell, cerquide,jar}@iiia.csic.es
[2] University of Southern California, Los Angeles, CA 90089
{eshieh, zhengyuy,tambe}@usc.edu
[3] University of the Pacific, Stockton, CA 95211
ebowring@pacific.edu

Abstract. Distributed constraint optimization (DCOP) is a promising approach to coordination, scheduling and task allocation in multi-agent networks. DCOP is NP-hard [6], so an important line of work focuses on developing fast incomplete solution algorithms that can provide guarantees on the quality of their local optimal solutions.
Region optimality [11] is a promising approach along this line: it provides quality guarantees for region optimal solutions, namely solutions that are optimal in a specific region of the DCOP. Region optimality generalises k- and t-optimality [7, 4] by allowing to explore the space of criteria that define regions to look for solutions with better quality guarantees.
Unfortunately, previous work in region-optimal quality guarantees fail to exploit any a-priori knowledge of the reward structure of the problem. This paper addresses this shortcoming by defining reward-dependent region optimal quality guarantees that exploit two different levels of knowledge about rewards, namely: (i) a ratio between the least minimum reward to the maximum reward among relations; and (ii) the minimum and maximum rewards per relation.

1 Introduction

Distributed Constraint Optimization (DCOP) is a popular framework for cooperative multi-agent decision making. It has been applied to real-world domains such as sensor networks [12], traffic control [3], or meeting scheduling [8]. In real-world domains, and particularly in large-scale applications, DCOP techniques have to cope with limitations on resources and time available for reasoning. Because DCOP is NP-Hard [6], complete DCOP algorithms (e.g. Adopt [6], OptAPO [5], DPOP [8]) that guarantee global optimality are unaffordable for these domains due to their exponential costs. In contrast to complete algorithms, incomplete algorithms [12, 2, 10, 7, 4] provide better scalability.

Unfortunately, an important limitation for the application of incomplete algorithms is that they usually fail to provide quality guarantees on their solutions.

The importance of quality guarantees is twofold. First, they help guarantee that agents do not converge to a solution whose quality is below a certain fraction of the optimal solution (which can have catastrophic effects in certain domains). Secondly, quality guarantees can aid in algorithm selection and network structure selection in situations where the algorithmic cost of coordination must be weighed up against solution quality (trade-off cost versus quality).

To the best of our knowledge, region optimal algorithms [11] are the only incomplete DCOP algorithms that can provide guarantees on the worst-case solution quality of their solutions at design time and exploit the available knowledge, if any, about the DCOP(s) to solve regarding their graph structure. The region optimal framework, that generalises the k- and t- optimal frameworks proposed in [7, 4], defines quality guarantees for region optimal solutions, namely solutions that are optimal in specific region of the DCOP. Thus, region optimality allows to explore the space of local optimality criteria (beyond size and distance) looking for those that lead to better solution qualities. To assess region optimal quality guarantees, in our previous work [11] we propose two methods with different computational costs: (1) a first one, based on solving an LP, that guarantees tightness; and (2) a second one that requires linear time but does not ensure tightness.

Unfortunately, previous work in region-optimal bounds fail to exploit some a-priori knowledge of the reward structure of the DCOP problem, if available, and only the knowledge of graph structure is exploited so far. As argued in [1] for the particular case of k-optimality, this limits the applicability of these approaches to many domains for which some information about the range of rewards is available. For example, in sensor networks [12], we may know the maximum reward of observing a phenomenon and the minimum reward when we have no observations.

This is the shortcoming we address in this paper and at this aim we extend the region optimal bounds to be able to exploit two different kinds of knowledge about the reward structure. Concretely, we show how to tight region optimal quality guarantees by assuming: (1) a ratio between the least minimum reward to the maximum reward among relations, along the lines of [1] (e.g. the ratio between the maximum reward of observing a phenomenon and the minimum reward when no observation is known); and (2) that the minimum and maximum rewards per relation are known (e.g. the maximum reward of observing a phenomenon and the minimum reward when no observation are known).

This paper is organised as follows. Section 2 provides some background on DCOPs and on the region optimal framework. Section 3 extends region optimal quality guarantees to exploit some a-priori knowledge about the reward structure. Section 4 analyses the tightness of the proposed reward-based guarantees and the improvement with respect to the guarantees formulated in [11]. Finally, section 5 draws conclusions.

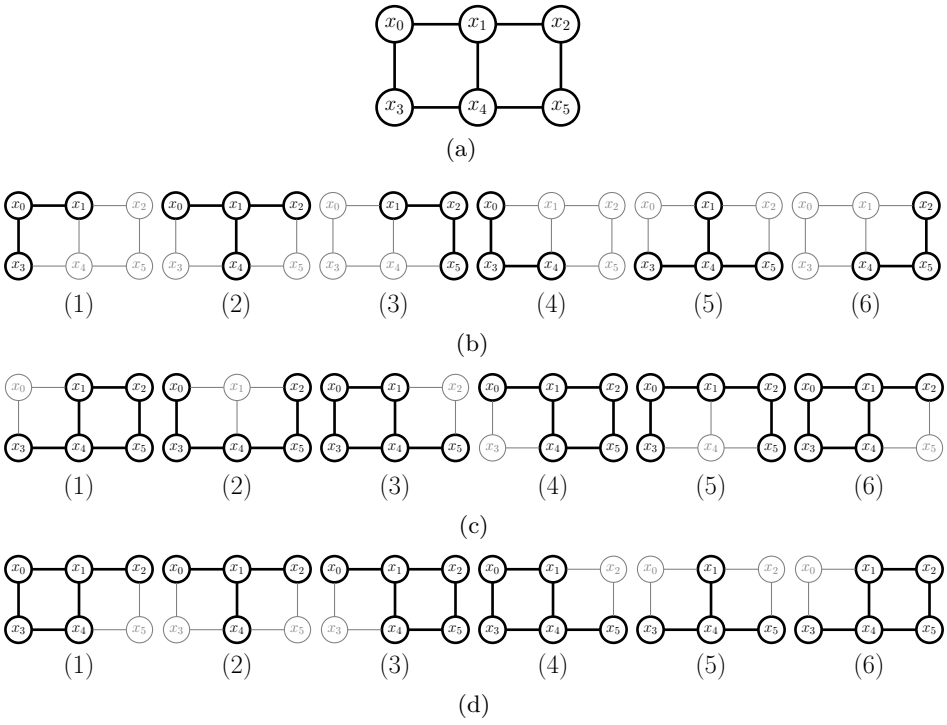


Fig. 1. Example of (a) a DCOP graph, (b) its 1-distance region, (c) its 5-size region and (d) its 5-size-distance-bounded region

2 Background

2.1 DCOP

A Distributed Constraint Optimization Problem (DCOP) consists of a set of variables, each assigned to an agent which must assign a discrete value to the variable: these values correspond to individual actions that can be taken by agents. Constraints exist between subsets of these variables that determine rewards to the agent team based on the combinations of values chosen by their respective agents, namely relations. Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. A relation on a set of variables $V \subseteq \mathcal{X}$ is expressed as a reward function $S_V : \mathcal{D}_V \rightarrow \mathbb{R}^+$, where $\mathcal{D}_V$ is the joint domain over the variables in V. This function represents the reward generated by the relation over the variables in V when the variables take on an assignment in the joint domain $\mathcal{D}_V$. Whenever there is no need to identify the domain, we simply use S to note relations.expressed as negative rewards).

In a DCOP each agent knows all the relations that involve its variable(s). In this work we assume that each agent is assigned a single variable, so we will use the terms "agent" and "variable" interchangeably.

Formally, a DCOP is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where: $\mathcal{X}$ is a set of variables (each one assigned to a different agent); $\mathcal{D}$ is the joint domain space for all variables; and $\mathcal{R}$ is a set of reward relations. The solution quality for an assignment $d \in \mathcal{D}$ to the variables in $\mathcal{X}$ is the sum of the rewards for the assignment over all the relations in the DCOP, namely:

$$R(d) = \sum_{S_V \in \mathcal{R}} S_V(d_V) \tag{1}$$

where $d_V \in \mathcal{D}_V$ contains the values assigned by d to the variables in V. With slight abuse of notation we allow to write equation 1 as $R(d) = \sum_{S \in \mathcal{R}} S(d)$.

Solving a DCOP amounts to choosing values for the variables in $\mathcal{X}$ such that the solution quality is maximized. A binary DCOP (each relation involves a maximum of two variables) is typically represented by its constraint graph, whose vertexes stand for variables and whose edges link variables that have some direct dependency (appear together in the domain of some relation). An example of a constraint graph is depicted in figure 1(a).

2.2 Region optimality

In [11] we introduce region optimality, a framework that generalise the k- and t-optimal frameworks [7, 4] by providing reward-independent quality guarantees for optima in regions characterised by any arbitrary criterion. Region optimality allows to explore the space of criteria (beyond size and distance) looking for those that lead to better solution qualities. Next we give a brief overview of the region optimal framework by defining the concepts of neighbourhood, region and region optimal solution.

Formally, a neighbourhood, A, is a subset of variables of $\mathcal{X}$. For instance, figure 1(b)(1) depicts a neighbourhood for the DCOP in figure 1(a) where boldfaced nodes in the constraint graph stand for variables included in the neighbourhood, namely $\{x_0, x_1, x_3\}$. Given two assignments x and y, we define $D(x, y)$ as the set containing the variables whose values in x and y differ. Then given a neighbourhood A, we say that x is a *neighbour of y in A* iff x differs from y only in variables that are contained in A, thus $D(x, y) \subseteq A$.

Given some assignment x, we say that it is optimal in a neighbourhood A if its reward cannot be improved by changing the values of some of the variables in the neighbourhood. That is, for every assignment y such that x is a *neighbour of y in A*, we have that $R(x) \geq R(y)$. Thus, an assignment x is optimal in the neighbourhood of figure 1(b)(1) if any other assignment that maintains the values of x_2, x_4 and x_5 as in x receives at most the same reward as x.

Given two neighbourhoods $A, B \subseteq \mathcal{X}$ we say that B completely covers A if $A \subseteq B$. We say that B does not cover A at all if $A \cap B = \emptyset$. Otherwise, we say that B covers A partially. For each neighbourhood A we can classify each relation S_V

in a DCOP into one of three disjoint groups, depending on whether C^α covers V completely ($T(A)$), partially ($P(A)$), or not at all ($N(A)$). For example, given the neighbourhood $\{x_0, x_1, x_3\}$ in figure 1(b)(1) we can classify the relations of the DCOP in figure 1(a) as : $T(\{x_0, x_1, x_3\}) = \{S_{\{x_0,x_3\}}, S_{\{x_0,x_1\}}\}$, $P(\{x_0, x_1, x_3\}) = \{S_{\{x_1,x_2\}}, S_{\{x_3,x_4\}}, S_{\{x_1,x_4\}}\}$, $N(\{x_0, x_1, x_3\}) = \{S_{\{x_2,x_5\}}, S_{\{x_4,x_5\}}\}$.

A region $\mathcal{C}$ is a multi-set[1] of subsets of $\mathcal{X}$, namely a multi-set of neighbourhoods of $\mathcal{X}$. For instance, figure 1(b) show a region composed of six neighbourhoods (b)(1)-(b)(6). Given a region $\mathcal{C}$, we say that x is *inside region* $\mathcal{C}$ *of* y iff x differs from y only in variables that are contained in one of the neighbourhoods in $\mathcal{C}$, that is, if there is a neighborhood $C^\alpha \in \mathcal{C}$ such that x is neighbour of y in C^α. Then, we can claim optimality for x in a region $\mathcal{C}$ (noted as $x^{\mathcal{C}}$) whenever it is optimal in each neighbourhood $C^\alpha \in \mathcal{C}$, that is if it cannot be improved by any other assignment inside region $\mathcal{C}$. For instance, an assignment x will be optimal in the region depicted in figure 1(a) if it is optimal in each of its six neighbourhoods.

Finally, for each relation $S_V \in \mathcal{R}$ we define $cc(S_V, \mathcal{C}) = |\{C^\alpha \in \mathcal{C} \text{ s.t } V \subseteq C^\alpha\}|$, that is, the number of neighbourhoods in $\mathcal{C}$ that cover the domain of S_V completely. We also define $nc(S_V, \mathcal{C}) = |\{C^\alpha \in \mathcal{C} \text{ s.t } V \cap C^\alpha = \emptyset\}|$, that is, the number of neighbourhoods in $\mathcal{C}$ that do not cover the domain of S_V at all. For instance, in the region of figure 1(b) there are two neighbourhoods that totally cover the relation $S_{\{x_0,x_1\}}$, namely neighbourhoods (1) and (2). Thus, $cc(S_{\{x_0,x_1\}}, \mathcal{C}) = 2$. Moreover, there is only one neighbourhood that do not cover $S_{\{x_0,x_1\}}$ at all, namely neighbourhood (6). Thus, $nc(S_{\{x_0,x_1\}}, \mathcal{C}) = 1$.

In [11] we show how region optimality generalises $k-$ and $t-$optimality by observing that: (i) both criteria are based on the definition of a region over the constraint graph; and (ii) given any assignment, checking for either k-size or t-distance optimality amounts to checking for optimality in that region. For example, figure 1(b) shows the neighbourhoods corresponding to the 1-distance region of the DCOP in figure 1(a), where each neighbourhood corresponds to one variable and its direct neighbours (e.g. neighbourhood (1) includes variable x_0 and its direct neighbours) whereas figure 1(c) shows the neighbourhoods corresponding to the 5-size region of the DCOP in figure 1(a), where each neighbourhood stands for a set of five connected variables.

Furthermore, region optimality allows to explore the space of arbitrary criteria to generate regions that otherwise will never be explored by size or distance criteria. For example, figure 1(d) shows a region created by the size-bounded-distance criteria proposed in [11]. Observe that the 5-size-bounded-distance region will never be created by either size or distance criteria because: (1) regarding size, it includes regions of size 4 and size 5; and (2) regarding distance, the region includes neighbourhoods different from the 1-distance region (shown in figure 1(b)) and from the 2-distance region which includes neighbourhoods that contain all the variables in the DCOP.

[1] A multi-set is a generalisation of a set that can hold multiple instances of the very same element.

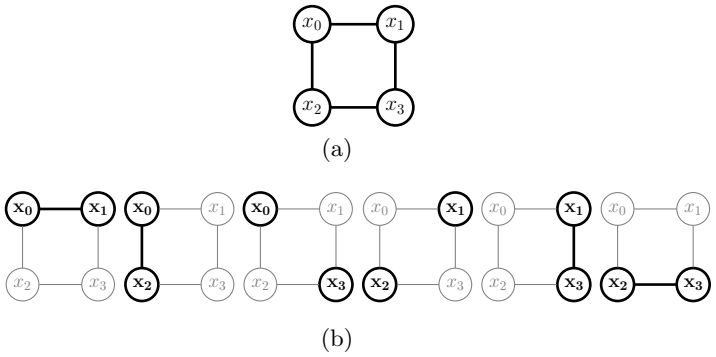


Fig. 2. Example of (a) a DCOP graph and (b) its 2-size region.

In the next section we describe the methods to calculate bounds for a $\mathcal{C}$-optimal assignment, namely an assignment that is optimal in an arbitrary region $\mathcal{C}$.

2.3 Region optimal reward-independent quality guarantees

In this section we review the methods proposed in [11] to calculate reward-independent quality guarantees for any region optima. In [11] we propose two methods to calculate region optimal reward-independent quality guarantees each one with a different computational cost: (1) a first one, based on solving an LP, that guarantees tightness; and (2) a second one, that requires linear time but does not ensure tightness. Assuming no knowledge of reward structure (except from they are non-negative) but exploiting the knowledge of the graph structure when available, these methods provide worst case quality of a $\mathcal{C}$-optimal solution as a fraction of the global optimal, where $\mathcal{C}$ is an arbitrary region.

We say that we have a bound δ when we can state that the quality of any $\mathcal{C}$-optimal assignment $x^{\mathcal{C}}$ is larger than δ times the quality of the optimal x^*. Hence, having a bound δ means that for every $x^{\mathcal{C}}$ we have that $\frac{R(x^{\mathcal{C}})}{R(x^*)} \geq \delta$.

Tight region optima quality guarantees. For a given set of relations $\mathcal{R}$, let $x^{\mathcal{C}}_{-}$ be the $\mathcal{C}$-optimal assignment with smallest reward, then $\frac{R(x^{\mathcal{C}}_{-})}{R(x^*)}$ provides a tight bound on the quality of any $\mathcal{C}$-optimal for the specific rewards $\mathcal{R}$.

In we show how to calculate a tight bound on a $\mathcal{C}$-optimal assignment independently from the specific DCOP rewards by directly searching the space of reward values to find the set of rewards $\mathcal{R}^*$ that minimizes $\frac{R^*(x^{\mathcal{C}}_{-})}{R^*(x^*)}$.

The assessment of this bound involves to solve the following program:

Find $\mathcal{R}$, $x^{\mathcal{C}}$ and x^* that
minimize $\frac{R(x^{\mathcal{C}})}{R(x^*)}$
subject to $x^{\mathcal{C}}$ being a $\mathcal{C}$-optimal for $\mathcal{R}$

Given the definition of region optimality, the condition of being a $\mathcal{C}$-optimal for $\mathcal{R}$ can be expressed as: for each x inside region $\mathcal{C}$ of $x^{\mathcal{C}}$ we have that $R(x^C) \geq R(x)$. However, instead of considering all the assignments for which x^C is guaranteed to be optimal, we consider only the subset of assignments such that the set of variables that deviate with respect to x^C take the same value than in the optimal assignment. If we restrict to this subset of assignments, then each neighbourhood covers a $2^{|C^\alpha|}$ assignments, one for each subset of variables in the neighbourhood. Let 2^{C^α} stand for the set of all subsets of the neighbourhood C^α. Then for each $A^k \in 2^{C^\alpha}$ we can define an assignment x^{α_k} such that for every variable x_i in a relation completely covered by A^k we have that $x_i^{\alpha_k} = x_i^*$, and for every variable x_i that is not covered at all by A^k we have that $x_i^{\alpha_k} = x_i^C$. Then, we can write the value of x^{α_k} as :

$$R(x^{\alpha_k}) = \sum_{S \in T(A^k)} S(x^*) + \sum_{S \in P(A^k)} S(x^{\alpha_k}) + \sum_{S \in N(A^k)} S(x^{\alpha_k}) \tag{2}$$

Now, the definition of $\mathcal{C}$-optimal can be expressed as:

$$R(x^C) \geq \sum_{S \in T(A^k)} S(x^*) + \sum_{S \in P(A^k)} S(x^{\alpha_k}) + \sum_{S \in N(A^k)} S(x^C) \quad \forall A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\} \tag{3}$$

that, by setting partially covered relations to the minimum possible reward (0 assuming non-negative rewards), results in:

$$R(x^C) \geq \sum_{S \in T(A^k)} S(x^*) + \sum_{S \in N(A^k)} S(x^C) \quad \forall A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\} \tag{4}$$

where $T(A^k)$ is the set of completely covered relations, $P(A^k)$ the set of partially covered relations and $N(A^k)$ the set of relations not covered at all.

Applying these transformations detailed in [9], we can simplify the initial program into the following linear program (LP) with z and y being vectors of positive real numbers:

minimize $\sum_{S \in \mathcal{R}} z_S$
subject to
$\sum_{S \in \mathcal{R}} y_S = 1$
and for each neighbourhood $A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\}$ covered by $\mathcal{C}$ subject to
$\sum_{S \in \mathcal{R}} z_S \geq \sum_{S \in T(A^k)} y_S + \sum_{S \in N(A^k)} z_S$

where $T(A^k)$ contains the relations completely covered by A^k and $N(A^k)$ the relations that are not covered by A^k at all.

As an example, consider the DCOP constraint graph in figure 2(a) and its 2-size region depicted in figure 2(b) for which we assess the LP region optimal

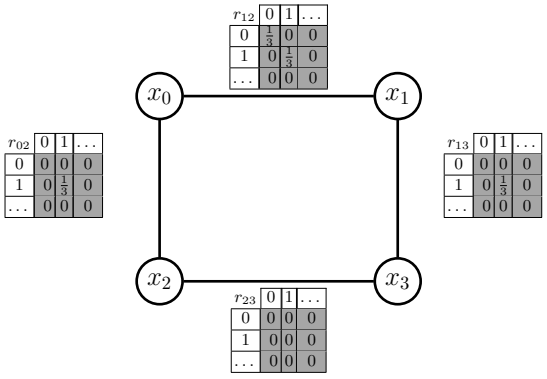


Fig. 3. Example of reward tables for which the 2-size region optimal bound for the DCOP of figure 2(a) is tight

bound as follows. In this case, we assume $x^{\mathcal{C}}_{-} = \langle x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0\rangle$ and $x^* = \langle x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 1\rangle$ where 0 and 1 stand for the first and second value in each variable domain. First we create the real variables, two for each of the four relations. Thus, given the relation $S_{x_0x_1}$ we create two real variables: one representing the value of $x^{\mathcal{C}}_{-}$, $z_{x_0x_1}$, and one representing the value of x^*, $y_{x_0x_1}$. Finally, to guarantee the optimality of $x^{\mathcal{C}}$ we add six constraints, one for each neighbourhood that compose the 2-size region depicted in figure 2(b). Thus, for the neighborhood depicted at the left of the figure 2(b) (composed of variables x_0, x_1), we constraint via $c0$ that the value of $x^{\mathcal{C}}_{-}$ must be greater than the sum of the values of totally covered relations for x^* ($y_{x_0x_1}$) plus the values of non-covered relations for $x^{\mathcal{C}}_{-}$ ($z_{x_2x_3}$). The resultant linear programming formulation is:

$$\begin{aligned}
&\text{minimize } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2}\\
&\text{subject to}\\
&\quad y_{x_0x_1} + y_{x_1x_3} + y_{x_2x_3} + y_{x_0x_2} = 1\\
&\text{and subject to:}\\
&\quad \text{(c0)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_0x_1} + z_{x_2x_3}\\
&\quad \text{(c1)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_0x_2} + z_{x_1x_3}\\
&\quad \text{(c2)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq 0\\
&\quad \text{(c3)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq 0\\
&\quad \text{(c4)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_1x_3} + z_{x_0x_2}\\
&\quad \text{(c5)}\ \ z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_2x_3} + z_{x_0x_1}
\end{aligned}$$

After solving this LP, $\delta = \sum_{S\in\mathcal{R}} z_S$ provides a tight bound on the quality of a $\mathcal{C}$-optimal solution for the graph structure represented by $\mathcal{R}$. Thus, by solving the LP for the 2-size optimal region in figure 2(b) we obtain a bound $\delta = \frac{1}{3}$. Moreover, we can use the values of the instantiated real variables, corresponding to the relations rewards for $x^{\mathcal{C}}_{-}$ and x^*, to generate DCOPs for which the assessed

bound is tight. Figure 2.3 shows a DCOP with a reward structure for which the 2-size region optimal bound $\delta = \frac{1}{3}$ obtained for the constraint graph in figure 2(a) is tight. It is easy to see that value of the 2-size optimal $x^{\mathcal{C}}_{-} = \langle 0, 0, 0, 0 \rangle$ is 1/3, higher than the value of any assignment inside the 2-size region, whereas the value of the optimal assignment $x^* = \langle 1, 1, 1, 1 \rangle$ is 1.

Let M be the number of variables of the largest neighbourhood in $\mathcal{C}$. The LP has $2 \cdot |\mathcal{R}|$ variables and $\mathcal{O}(2^M \cdot |\mathcal{C}|)$ constraints, and hence it is solvable in time polynomial in $|\mathcal{R}|$ and in $2^M \cdot |\mathcal{C}|$.

Faster quality guarantees for region optima. Because the computational complexity of the LP method can be high as the number of relations $|\mathcal{R}|$, the number of neighbourhoods $|\mathcal{C}|$ or its size M grows in we propose a faster alternative method to compute region optimal bounds. This faster method can compute a bound, by means of proposition 1, in time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$ but, as a counterpart, we lose the tightness of the bound.

Proposition 1. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP with non-negative rewards and $\mathcal{C}$ a region. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then*

$$R(x^{\mathcal{C}}) \geq \frac{cc_*}{|\mathcal{C}| - nc_*} R(x^*) \tag{5}$$

where $cc_* = \min_{S \in \mathcal{R}} cc(S, \mathcal{C})$, $nc_* = \min_{S \in \mathcal{R}} nc(S, \mathcal{C})$, and x^* is the optimal assignment.

Proposition 1 proved in [11] directly provides a simple algorithm to compute a bound. Given a region $\mathcal{C}$ and a graph structure, we can directly assess cc_* and nc_* by computing $cc(S, \mathcal{C})$ and $nc(S, \mathcal{C})$ for each relation $S \in \mathcal{R}$ and taking the minimum. This will take time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$, that is linear in the number of relations of the DCOP and linear in the number of neighbourhoods in the region.

Despite its complexity improvements, the bound assessed by proposition 1 is not guaranteed to be tight and can return worse bounds than the ones provided by the LP-based mechanism. As an example, now we turn back to figure 2 to assess the bounds for the 2-size optimal region in figure 2(b) using equation 5. Given the relation $S_{\{x_0, x_1\}}$, we assess the number of neighbourhoods that completely cover $\{x_0, x_1\}$ as $cc(S_{\{x_0,x_1\}}, \mathcal{C}_2) = 2$ (the first neighbourhood) and the number of neighbourhoods that do not cover $\{x_0, x_1\}$ at all as $nc(S_{\{x_0,x_1\}}, \mathcal{C}_2) = 1$ (the sixth neighbourhood). After repeating the process for the rest of relations in the constraint graph, we obtain that $cc_* = 1$ and $nc_* = 1$, and hence $\frac{cc_*}{|\mathcal{C}_1| - nc_*} = \frac{2}{6-2} = \frac{1}{2}$ and hence, this faster bound is not tight (compare with the tight bound assessed above $\delta = \frac{1}{3}$).

Both the LP and proposition 1 assess bounds that depend on the graph structure but are independent of the specific reward values. We can always use them to assess bounds independently of the graph structure by assessing the bound for the complete graph, since any other structure is a particular case of the complete graph with some rewards set to zero.

3 Reward-based region optimal bounds

Previous section reviewed two methods that provide bounds on any $\mathcal{C}$-optimal, characterized by an arbitrary $\mathcal{C}$ criterion, that are independent of the specific reward structure. However, as shown in [1] for the specific criterion of group size, if some knowledge of the reward structure of the problem is available then it can be exploited to assess more accurate bounds. Here we show how to incorporate reward structure knowledge in the region optimality framework and formulate two different improvements by assuming:

- a ratio between the least minimum reward to the maximum reward among constraints, the so-called *minimum fraction reward* (section 3.1); and
- the knowledge of the minimum and maximum rewards per relation, the so-called *extreme relations rewards* (section 3.2).

Finally, section 4 provides results to characterise the gain on tightness obtained when exploiting the knowledge about these different reward structures.

3.1 Based on the minimum fraction reward

In this section we show how to tight the LP-based or the faster region optimal bounds described in section 2.3 when we know that the minimum reward is a certain factor β ($0 < \beta \leq 1$) of the maximum reward on any relation. Thus, this refinement is a generalization of the improvements in tightness for size-optimal bounds proposed in [1].

Extending the LP-based mechanism to exploit knowledge of the minimum fraction reward First, we show how assuming a minimum fraction reward of β we tight the quality guarantees obtained by means of the LP mechanism described in section 2.3. In order to obtain a tighter bound, we will employ the set of partially covered relations.

In this case, instead of setting the values of all relations $\sum_{S \in P(C^{\alpha})} S(x^{\alpha_k})$ to 0, as in equation 4, we can exploit the knowledge that $S(x^{\alpha_k}) \geq \beta \cdot S(x^*)\ \forall S \in \mathcal{R}$. Then, $\forall A^k \in \{2^{C^{\alpha_k}} | C^{\alpha_k} \in \mathcal{C}\}$, the restriction for assignment x^{α_k} is rewritten as:

$$\sum_{S \in \mathcal{R}} S(x^C) \geq \sum_{S \in T(C^{\alpha})} S(x^*) + \sum_{S \in P(C^{\alpha})} \beta \cdot S(x^*) + \sum_{S \in N(C^{\alpha})} S(x^C) \tag{6}$$

Notice that it is the only change we need to do to incorporate the knowledge of the minimum fraction reward and that the program can be simplified to an LP, following analogous operations as the ones detailed in [9], with the same number of variables than in the reward independent case.

As an example, we turn back to figure 2 to assess the LP region optimal bound for the 2-size region depicted in figure 2(b) when assuming a minimum fraction reward β. With respect to the reward-independent LP formulation, the right part of each constraint is modified to add the real variables related to the

values of x^* for the partially covered relations multiplied by β. This results in the following LP formulation:

$$
\begin{aligned}
&\text{minimize } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \\
&\text{subject to} \\
&\quad y_{x_0x_1} + y_{x_1x_3} + y_{x_2x_3} + y_{x_0x_2} = 1 \\
&\text{and subject to:} \\
&\quad \text{(c0) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_0x_1} + \beta \cdot (y_{x_1x_3} + y_{x_0x_2}) + z_{x_2x_3} \\
&\quad \text{(c1) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_0x_2} + \beta \cdot (y_{x_0x_1} + y_{x_2x_3}) + z_{x_1x_3} \\
&\quad \text{(c2) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq \beta \cdot (y_{x_0x_1} + y_{x_1x_3} + y_{x_2x_3} + y_{x_0x_2}) \\
&\quad \text{(c3) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq \beta \cdot (y_{x_0x_1} + y_{x_1x_3} + y_{x_2x_3} + y_{x_0x_2}) \\
&\quad \text{(c4) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_1x_3} + \beta \cdot (y_{x_0x_1} + y_{x_2x_3}) + z_{x_0x_2} \\
&\quad \text{(c5) } z_{x_0x_1} + z_{x_1x_3} + z_{x_2x_3} + z_{x_0x_2} \geq y_{x_2x_3} + \beta \cdot (y_{x_1x_3} + y_{x_0x_2}) + z_{x_0x_1}
\end{aligned}
$$

The solution of the LP defines a tight bound on the quality of a $\mathcal{C}$-optimal solution for the graph structure represented by $\mathcal{R}$ and rewards with a minimum fraction reward of β. Thus, by solving the above LP with β set to $\frac{1}{2}$ we assess a $\mathcal{C}$-optimal bound $\delta = \frac{2}{3}$ for DCOPs with rewards with a minimum fraction reward of $\frac{1}{2}$. Notice that this per-reward bound provides a significant increment with respect to the reward-independent bound assessed in section 2.3 and with respect to the straightforward bound we can obtain by only taking into account β, namely $\delta = \frac{1}{2}$.

Extending the faster mechanism to exploit knowledge of the minimum fraction reward Second, we show how assuming a minimum fraction reward of β we can improve the faster $\mathcal{C}$-optimal bounds, reviewed in section 2.3. Before that we define, for each relation $S \in \mathcal{R}$, $pc(S, \mathcal{C}) = |\mathcal{C}| - nc(S, \mathcal{C}) - cc(S, \mathcal{C})$ as the number of neighbourhoods in region $\mathcal{C}$ that partially cover relation S. Then, the following proposition shows how to exploit the minimum fraction reward along with the partially covered relations to obtain a bound tighter than the one in equation 5.

Proposition 2. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region and β the minimum fraction reward. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \left(\frac{cc_*}{|\mathcal{C}| - nc_*} + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \right) \mathcal{R}(x^*) \tag{7}$$

where $cc_* = \min_{S \in \mathcal{R}} cc(S, \mathcal{C})$, $nc_* = \min_{S \in \mathcal{R}} nc(S, \mathcal{C})$, $pc_* = \min_{S \in \mathcal{R}} pc(S, \mathcal{C})$, and x^* is the optimal assignment.

Proposition 2 directly provides a simple algorithm to compute a bound. Given a region $\mathcal{C}$ and a graph structure, we can directly assess cc_*, pc_* and nc_* by computing $cc(S, \mathcal{C})$, $pc(S, \mathcal{C})$ and $nc(S, \mathcal{C})$ for each relation $S \in \mathcal{R}$ and taking the minimum. This will take time $\mathcal{O}(|\mathcal{R}||\mathcal{C}|)$, that is linear in the number of relations of the DCOP and linear in the number of neighbourhoods in the region. As an

example, now we turn back to figure 2 to assess the bounds for the 2-size region $\mathcal{C}_2$ in figure 2(b) when assuming a minimum fraction reward $\beta = \frac{1}{2}$. Given the relation S_{x_0,x_1} we assess the number of neighbourhoods that completely cover $\{x_0, x_1\}$ as $cc(S_{x_0x_1}, \mathcal{C}_2) = 1$ (the first neighbourhood), the number of neighbourhoods that partially cover $\{x_0, x_1\}$ as $pc(S_{x_0x_1}, \mathcal{C}_2) = 4$ (from the second to the fifth neighborhoods) and the number of neighbourhoods that do not cover $\{x_0, x_1\}$ at all as $nc(S_{x_0x_1}, \mathcal{C}_2) = 1$ (the sixth neighbourhood). After repeating the process for the rest of relations in the constraint graph, we obtain that $cc_* = 1$, $pc_* = 4$ and $nc_* = 1$, and hence $\frac{cc_*}{|\mathcal{C}|-nc_*} + \beta\frac{pc_*}{|\mathcal{C}|-nc_*} = \frac{1}{6-1} + \frac{1}{2}\cdot\frac{4}{6-1} = \frac{3}{5}$. Notice that this leads to a significant improvement with respect to the reward-independent faster bound assessed in section 2.3 as $\delta = \frac{1}{5}$ although this bound is also not tight (compare with the $\mathcal{C}$-optimal bound $\delta = \frac{2}{3}$ assessed by means of the LP).

Proof. The proof is analogous to the one for the general bound of equation 5 formulated in [11], but without disregarding partially covered relations. For every $C^\alpha \in \mathcal{C}$, consider an assignment x^α such that: $x_i^\alpha = x_i^{\mathcal{C}}$ if $x_i \notin C^\alpha$, and $x_i^\alpha = x_i^*$ if $x_i \in C^\alpha$. Since $x^{\mathcal{C}}$ is $\mathcal{C}$-optimal, for all $C^\alpha \in \mathcal{C}$, $\mathcal{R}(x^{\mathcal{C}}) \geq \mathcal{R}(x^\alpha)$ holds, and hence:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \frac{\sum\limits_{C^\alpha \in \mathcal{C}} \left(\sum\limits_{S \in T(C^\alpha)} S(x^*) + \sum\limits_{S \in N(C^\alpha)} S(x^{\mathcal{C}}) + \sum\limits_{S \in P(C^\alpha)} S(x^\alpha) \right)}{|\mathcal{C}|}. \tag{8}$$

Using β we can express the third term in equation 8 in terms of $\mathcal{R}(x^*)$ considering that the knowledge for any relation $S \in \mathcal{R}$ satisfies $S(x^\alpha) \geq \beta \cdot S(x^*)$. Therefore, the following inequalities hold:

$$\sum_{C^\alpha \in \mathcal{C}} \sum_{S \in P(C^\alpha)} S(x^\alpha) \geq \sum_{S \in \mathcal{R}} pc(S, \mathcal{C}) \cdot \beta \cdot S(x^*) \geq pc_* \cdot \beta \cdot \mathcal{R}(x^*) \tag{9}$$

From the proof of the general bound of equation 5 in [11] we know that the first and the second sets of relations of equation 8 can be also expressed in terms of $\mathcal{R}(x^{\mathcal{C}})$ and $\mathcal{R}(x^*)$. Therefore, after substituting these results in equation 8 and rearranging terms, we obtain equation 7.

3.2 Based on the extreme relation rewards

In this section we show how to tight the reward-independent $\mathcal{C}$-optimal bound δ for a DCOP problem when we know the values of the minimum and maximum rewards for each relation $S \in \mathcal{R}$, namely $l_S = \min_{d_V \in \mathcal{D}_V} S(d_V)$ and $u_S = \max_{d_V \in \mathcal{D}_V} S(d_V)$ respectively.

Proposition 3. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP and $\mathcal{C}$ a region. If $x^{\mathcal{C}}$ is a $\mathcal{C}$-optimal assignment then:*

$$\mathcal{R}(x^{\mathcal{C}}) \geq \frac{1}{U}\left((U - L) \cdot \delta + L\right) \cdot \mathcal{R}(x^*) \tag{10}$$

where $U = \sum_{S \in \mathcal{R}} u_S$, $L = \sum_{S \in \mathcal{R}} l_S$.

Proposition 3 directly provides a constant-time method to tight any reward-independent $\mathcal{C}$-optimal bound δ by assuming that the extreme values of relations are known. Because proposition 3 does not make any assumption about how δ is calculated rather than it is reward-independent $\mathcal{C}$-optimal bound for $x^{\mathcal{C}}$, this improvement applies to both mechanism reviewed in section 2.3: (i) to the reward-independent $\mathcal{C}$-optimal bounds obtained by means of the LP; and (ii) to the faster reward-independent bounds of equation 8.

As an example we turn back to figure 2 to assess the bounds for the 2-size region $\mathcal{C}_2$ in figure 2(b) when assuming some particular extreme rewards per relation. First, assume a first scenario in which each relation of the DCOP in figure 2(a) has a minimum reward of 2 and a maximum reward of 4. Thus, $U = 16$ and $L = 8$. In this scenario, when the reward-independent bound is assessed by means of the LP method explained in section 2.3 $\delta = \frac{1}{3}$, and hence the bound of proposition 3 is $\frac{1}{U}\left((U - L) \cdot \delta + L\right) = 1/16 \cdot ((16 - 8) \cdot 1/3 + 8) = 2/3$. In a similar way, if δ is set to the faster reward-independent bound obtained by means of equation 5 $\delta = 1/5$, the bound of proposition 3 is $1/16 \cdot ((16-8) \cdot \frac{1}{5} + 8) = 3/5$. It is worth noting that these bounds are the same than the bounds obtained in the above section by the respective LP and the faster mechanisms when assuming a minimum fraction reward of $\beta = \frac{1}{2}$. Thus, in this first scenario the two different assumptions about the reward structure lead to the same bounds.

Now assume a second scenario in which the DCOP in figure 2(a) has two relations with a minimum reward of 2 and a maximum of 4, and two relations with a minimum reward of 3 and a maximum of 4. Thus, $U = 16$ and $L = 10$. In this scenario, when reward-independent bound is assessed by means of the LP method the bound of proposition 3 is $1/16 \cdot ((16 - 10) \cdot 1/3 + 10) = 3/4$ whereas when it is assessed by means of the faster method is $1/16 \cdot ((16 - 10) \cdot \frac{1}{5} + 10) = 7/10$. Therefore, in this second scenario, exploiting the knowledge about the extreme rewards of relations leads to tighter bounds than exploiting the knowledge about the minimum fraction reward.

Next we provide the proof for proposition 3.

Proof. Let $\hat{\mathcal{R}}$ be a distribution defined as $\hat{\mathcal{R}}(x) = \mathcal{R}(x) - L = \sum_{S \in \mathcal{R}}(S(x) - l_S)$. Notice that the rewards of $\hat{\mathcal{R}}$ are non-negative because after substracting the minimum of each non-negative relation of $\mathcal{R}$ we obtain new relations in which the minimum value is 0. Moreover, because the value of any assignment in $\hat{\mathcal{R}}$ is equal to the value in $\mathcal{R}$ plus a constant, any $\mathcal{C}$-optimal $x^{\mathcal{C}}$ in $\mathcal{R}$ is also $\mathcal{C}$-optimal in $\hat{\mathcal{R}}$. Thus, by definition of $\mathcal{C}$-optimal bounds the following inequality holds:

$$\hat{\mathcal{R}}(x^{\mathcal{C}}) \geq \delta \cdot \hat{\mathcal{R}}(x^*) \tag{11}$$

Then by expressing $\hat{\mathcal{R}}$ in terms of $\mathcal{R}$ and isolating $\mathcal{R}(x^{\mathcal{C}})$ we obtain:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \delta \cdot \mathcal{R}(x^*) + (1 - \delta) \cdot L \tag{12}$$

Now multiplying and dividing the right equation side by $\mathcal{R}(x^*)$:

$$\mathcal{R}(x^{\mathcal{C}}) \geq \left(\frac{\delta \cdot \mathcal{R}(x^*) + (1 - \delta) \cdot L}{\mathcal{R}(x^*)}\right) \cdot \mathcal{R}(x^*) \tag{13}$$

Since the bound provided by equation 13 above increases as the value of the optimum, $\mathcal{R}(x^*)$, decreases, we can get rid of $\mathcal{R}(x^*)$, which is in general unknown, by replacing it with an upper bound. By definition, U is an upper bound of $\mathcal{R}(x^*)$. Hence, we can substitute U for $\mathcal{R}(x^*)$ to obtain equation 10.

Equation 10 places the reward-independent bound in equation 5 within the $[L, U]$ scale, and subsequently assesses the fraction of upper bound U that it represents.

4 Comparing reward-dependent bounds

Since the more knowledge quality guarantees exploit from the problem the tighter they are likely to be, per-reward region optimal guarantees proposed in the above sections are expected to be tighter than reward-independent bounds of section 2.2. Furthermore, because not all the assumptions over the reward structure have the same level of specificity, exploiting the knowledge about the extreme rewards per relation is also expected to lead to tighter quality guarantees than only assuming a ratio between them.

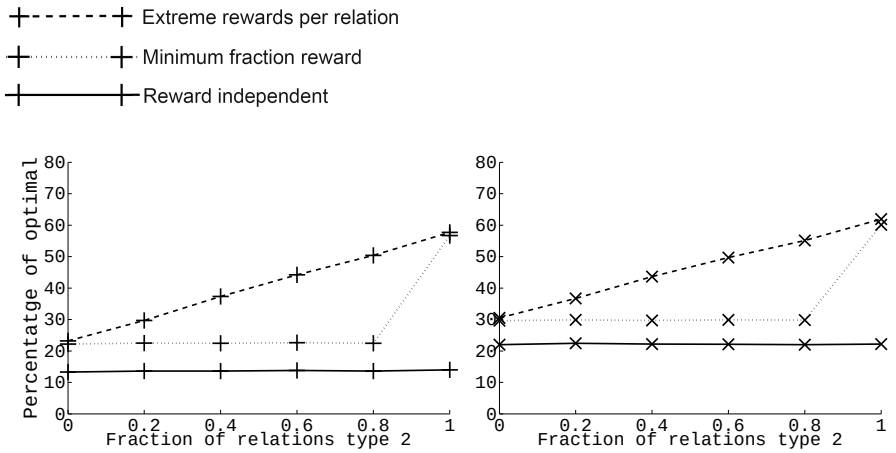
Hence, at the aim of illustrating the tightness of region optimal quality guarantees when exploiting different knowledge about the reward structure, we provide with: (i) empirical results that show the average-case improvement over LP-based region optimal bounds; and (ii) theoretical results that characterise the relations between faster region optimal bounds.

4.1 Comparing LP region optimal quality guarantees

Next we provide with results that illustrate the average-case region optimal bounds assessed with the LP mechanism.

Figures 4(a)(b) show the values of the region optimal bounds, defined as a percentage of the optimal, for random DCOPs with 100 agents and density 4 using as a criterion neighborhoods of size 3 and of distance 1 respectively. All results are averaged over 50 sample instances. Bounds are calculated using the LP method. Because, intuitively, the gain obtained by exploiting the knowledge about the extreme rewards per relation with respect to the minimum fraction reward varies with the heterogeneity of the reward structure, we generate DCOPs with two types of relations: (1) type 1, relations whose rewards are integers drawn from a uniform distribution $U[2500, 10000]$ and (2) type 2, relations whose rewards are integers drawn from a uniform distribution $U[5000, 10000]$.

The dotted lines show the per-reward bounds when exploiting the knowledge of the minimum fraction reward, the dashed lines when exploiting the knowledge of the extreme relation rewards and the solid lines show the region optimal bounds as presented in section 2.3, that apply to any reward structure. The x-axis represents the fraction of the total relations of type 2 with respect to the default relations of type 1. First of all observe that, independently of the particular knowledge exploited, per-reward bounds provide significant improvements with



(b) Size 3 bounds (c) Distance 1 bounds

Fig. 4. Reward-based bounds on 100 agent random DCOPs with density 4 using as a criterion (a) node size 3 and (b) distance 1.

respect to the reward-independent bound. For example, in figure 4 (a) the reward independent bound is around 12% whereas per-reward bounds are around 22% when exclusively relations of type 1 (x-axis= 0) and around 60% when exclusively using relations of type 2 (x-axis= 1). Moreover, it is worth noting that when all relations are of the same type, independently of the knowledge exploited about the reward, both per-reward bounds are very close. In contrast, in mixed instances, when both type of relations are present, graphs show that minimum fraction reward bounds can not improve the bound further by taking advantage of relations of type 2. Indeed, when exploiting the minimum fraction reward the LP mechanism does not get a significant improvement with the introduction of relations of type 2 until relations of type 1 disappear. In contrast, when exploiting the knowledge about the extreme relation rewards the region optimal bound progressively increases with the fraction of relations of type 2.

In summary, these results show how exploiting more knowledge about the reward structure, such as the extreme rewards per relation, help obtain tighter bounds for a wider range of reward distributions, particularly in heterogeneous distributions composed of rewards of different kind.

4.2 Comparing faster region optimal quality guarantees

In what follows we provide with some theoretical results on the improvement on bound tightness of the faster region optimal quality guarantees under the different reward structure assumptions.

On the one hand, it is easy to see that the faster per-reward region optimal guarantees assessed by equation 7 and 10 are guaranteed to be greater or, in the

worst-case equal, than the faster reward-independent region optimal guarantees assessed by means of equation 5.

On the other hand, we are interested in comparing per-reward faster quality guarantees when exploiting different assumptions over the reward structure. At this aim, next we prove that faster quality guarantees that exploit the knowledge about the extreme rewards per relation are guarantee to be tighter than those that exploit the minimum fraction bound.

Proposition 4. *Let $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a DCOP, $\mathcal{C}$ a region, β the minimum fraction reward, where $l_{S_V} = \min_{d_V \in \mathcal{D}_V} S_V(d_V)$, $u_{S_V} = \max_{d_V \in \mathcal{D}_V} S(d_V)$:*

$$\frac{cc_*}{|\mathcal{C}| - nc_*} + \beta \frac{pc_*}{|\mathcal{C}| - nc_*} \leq \frac{U - L}{U} \frac{cc_*}{|\mathcal{C}| - nc_*} + \frac{L}{U} \tag{14}$$

Proof. After rearranging terms and simplifying, we obtain that equation 14 is equivalent to:

$$pc_* \cdot \beta \leq (|\mathcal{C}| - nc_* - cc_*) \cdot \frac{L}{U} \tag{15}$$

First, from the definition of partial covering, $pc(S, C) = |\mathcal{C}| - nc(S, C) - cc(S, C)$, we observe that $pc(S, C)$ increases as $nc(S, C)$ and $cc(S, C)$ decrease. Since nc_*, cc_* are the minimum values that functions nc, cc can take on respectively, then $pc_* \leq |\mathcal{C}| - nc_* - cc_*$ holds. Therefore, proving that $\beta \leq \frac{L}{U}$, namely that $min_{S \in \mathcal{R}} \frac{l_S}{u_S} \leq \frac{\sum_{S \in \mathcal{R}} l_S}{\sum_{S \in \mathcal{R}} u_S}$, is enough to prove that equation 15 holds. We build the proof by induction of the number of relations $n = |\mathcal{X}|$. Consider without loss of generality a problem with n relations such that $\frac{l_{r_1}}{u_{r_1}} \leq \ldots \leq \frac{l_{r_{n-1}}}{u_{r_{n-1}}} \leq \frac{l_{r_n}}{u_{r_n}}$ holds. If $n = 2$, then $min(\frac{l_{r_1}}{u_{r_1}}, \frac{l_{r_2}}{u_{r_2}}) \leq \frac{l_{r_1} + l_{r_2}}{u_{r_1} + u_{r_2}}$ simplifies to $\frac{l_{r_1}}{u_{r_1}} \leq \frac{l_{r_2}}{u_{r_2}}$, which by problem definition is true. When $n > 2$ we must prove that $\frac{l_{r_1}}{u_{r_1}} \leq \frac{l_{r_1} + \sum_{j \geq 2} l_{r_j}}{u_{r_1} + \sum_{j \geq 2} u_{r_j}}$ holds, or equivalently that $\frac{l_{r_1}}{u_{r_1}} \leq \frac{\sum_{j \geq 2} l_{r_j}}{\sum_{j \geq 2} u_{r_j}}$. By recursively applying the expression for $n = 2$ to $\frac{\sum_{j \geq 2} l_{r_j}}{\sum_{j \geq 2} u_{r_j}}$ we obtain that:

$\frac{\sum_{j \geq 2} l_{r_j}}{\sum_{j \geq 2} u_{r_j}} = \frac{l_{r_2}}{u_{r_2}} + \frac{\sum_{j \geq 3} l_{r_j}}{\sum_{j \geq 3} u_{r_j}} \geq \min(\frac{l_{r_2}}{u_{r_2}}, \frac{\sum_{j \geq 3} l_{r_j}}{\sum_{j \geq 3} u_{r_j}}) \geq \ldots$

$\geq \min(\frac{l_{r_2}}{u_{r_2}}, \min(\frac{l_{r_3}}{u_{r_3}}, \ldots, \min(\frac{l_{r_{n-1}}}{u_{r_{n-1}}}, min(\frac{l_{r_n}}{u_{r_n}}) \ldots) = \frac{l_{r_2}}{u_{r_2}}$.

Thus, $\frac{\sum_{j \geq 2} l_{r_j}}{\sum_{j \geq 2} u_{r_j}} \geq \frac{l_{r_2}}{u_{r_2}}$ and consequently, $\frac{\sum_{j \geq 2} l_{r_j}}{\sum_{j \geq 2} u_{r_j}} \geq \frac{l_{r_1}}{u_{r_1}}$ holds.

5 Conclusions

In this paper we extended the region optimal region bounds in [11] to exploit some a-priori knowledge of the reward structure of the problem, if available. To that end, this paper provided reward-dependent bounds that incorporate as available prior knowledge: (i) a ratio between the least minimum reward to the maximum reward among relations; (ii) a minimum and maximum rewards per relation.

References

1. Bowring, E., Pearce, J.P., Portway, C., Jain, M., Tambe, M.: On k -optimal distributed constraint optimization algorithms: new bounds and algorithms. In: AAMAS (2). pp. 607–614 (2008)
2. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS (2). pp. 639–646 (2008)
3. Junges, R., Bazzan, A.L.C.: Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: AAMAS. pp. 599–606 (2008)
4. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: AAMAS. pp. 133–140 (2010)
5. Mailler, R., Lesser, V.R.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS. pp. 438–445 (2004)
6. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees. Artif. Intell. 161(1-2), 149–180 (2005)
7. Pearce, J.P., Tambe, M.: Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: IJCAI. pp. 1446–1451 (2007)
8. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: IJCAI. pp. 266–271 (2005)
9. Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J.A., Yin, Z., Tambe, M., Bowring, E.: LP formulation of regional-optimal bounds. Tech. Rep. TR-IIIA-2011-01 (2011), at: http://www.iiia.csic.es/files/pdfs/TR201101.pdf
10. Vinyals, M., Pujol, M., Rodriguez-Aguilar, J., Cerquides, J.: Divide and Coordinate: solving DCOPs by agreement. In: AAMAS. pp. 150–156 (2010)
11. Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J.A., Yin, Z., Tambe, M., Bowring, E.: Quality guarantees for region optimal algorithms. In: AAMAS (2011)
12. Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artif. Intell. 161(1-2), 55–87 (2005)

Including Human Behavior in Stackelberg Game for Security

Rong Yang[1], Christopher Kiekintveld[2],
Fernando Ordonez[1], Milind Tambe[1], and Richard John[1]

[1] University of Southern California
[2] University of Texas El Paso

Abstract. Recently, Stackelberg games have garnered significant attention given their deployment for real world security. However, a fundamental challenge of applying game-theoretic techniques to real-world security problem is the standard assumption that the adversary is perfectly rational in responding to security force's strategy, which can be unrealistic for human adversaries. Previous work has presented COBRA as a leading contender for accounting for the bounded rationality of human adversaries in security games. This paper presents an advance over this previous work by providing new algorithms based on two human behavior theories: Prospect Theory (PT) and Quantal Response Equilibrium (QRE). The paper's key contributions include: (i) efficient algorithms for computing optimal strategic solutions using PT and QRE; (ii) most comprehensive experiment to date on effectiveness of different models against human subjects; (iii) new techniques for generating representative payoff structures for behavioral experiments in generic classes of games. Our results with human subjects show that our new strategies significantly outperform COBRA.

Keywords: Human Behavior, Stackelberg Games, Decision-making

1 Introduction

Game-theoretic models have recently become important tools for analyzing real-world security resource allocation problems. These models provide a sophisticated approach for generating randomized strategies that mitigate attackers' ability to find weaknesses using surveillance. The ARMOR system at LAX airport [9] and IRIS at the Federal Air Marshals Service [13] are notable real-world deployments of this approach. One of the key sets of assumptions these systems make is about how attackers choose attack strategies based on their knowledge of the security policy. Typically, such systems have applied the standard game-theoretic assumption that attackers are perfectly rational and will strictly maximize their expected utility. This is a reasonable proxy for the worst case of a highly intelligent attacker, but it leaves open the possibility that the defender's strategy is not robust against attackers using different decision procedures, and

it fails to exploit known weaknesses in the decision-making of human attackers. Indeed, it is widely accepted that standard game-theoretic assumptions of perfect rationality are not ideal for predicting the behavior of humans in multi-agent decision problems [1]. In the multi-agent systems community there is a growing interest in adopting these models to improve decisions in agents that interact with humans or to provide better decision support in systems that use multi-agent systems techniques to provide advice to human decision-makers [2,3]. Our work in this paper focuses on integrating more realistic models of human behavior into the computational analysis of security problems.

There are several challenges in moving beyond perfect rationality assumptions to integrate more realistic models of human decision-making. First, the literature has introduced a multitude of candidate models, but there is an important empirical question of which model best represents the salient features of human behavior in applied security games. Second, integrating any of the proposed models into a decision-support system (even for the purpose of empirically evaluating the model) requires developing new methods for computing solutions to security games, since the existing algorithms are based on mathematically optimal attackers [6,8]. In this context, COBRA (Combined Observability and Rationality Assumption), developed in most recent work [10] is the leading contender that accounts for human behavior in security games. Thus, the open question is whether there are other approaches that allow for fast solution and yet outperform COBRA in addressing human behaviors.

This paper addresses the challenges and answers the open questions: it develops two new methods for generating defender strategies in security games based on using two well-known models of human behavior to model the attacker's decisions. The first is *Prospect Theory* (PT), which provides a descriptive framework for decision-making under uncertainty that accounts for both risk preferences and variations in how humans interpret probabilities through a weighting function [5]. The second model is Quantal Response Equilibrium (QRE). QRE adapts ideas from the literature on discrete choice problems to a game-theoretic framework with the basic premise that humans will choose better actions more frequently, but with some noise in the decision-making process that leads to stochastic choice probabilities following a logit distribution. We develop new techniques to compute optimal defender strategies in Stackelberg security games under the assumption that the attacker will make choices according to either the PT or QRE model.

To test these new methods we performed experiments with human subjects using an online game called 'The Guard and the Treasure' designed to simulate a security scenario similar to the ARMOR program for the Los Angeles International (LAX) airport. Furthermore, we designed classification techniques to select payoff structures for experiments such that the models are well separated from each other and the payoff structures are representative of the game space. We compare these models against both a perfect rationality baseline (DOBSS) and COBRA. Our data shows that the new approaches yield statistically significantly better strategies against human attackers than previous methods in-

cluding COBRA in most of the payoff structures, and comparable results in others.

2 Stackelberg Security Game

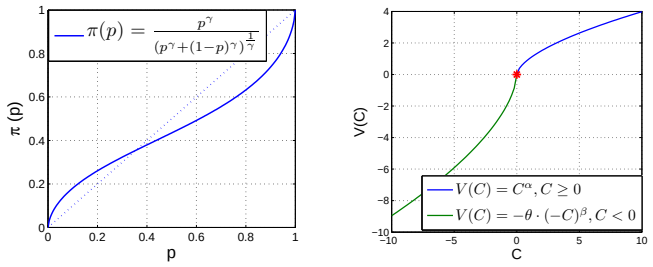
We consider a Stackelberg game with a single leader and at least one follower. The leader commits to a strategy first, taking into account the follower's response to her strategy. The followers decide their actions knowing the leader strategy. Security games refer to a special class of attacker-defender Stackelberg games, used in deployed applications mentioned earlier [9,13], where the defender plays the role of leader and an adversary plays the role of follower. In these non zero-sum games the attacker's utility of attacking a target decreases as the defender allocates more resources to protect it (and vice versa for the defender). The defender (leader) first commits to a mixed strategy, assuming the attacker (follower) decides on a pure strategy after observing the defender's strategy. This models the situation where an attacker conducts surveillance to learn the defender's mixed strategy and then launches an attack on a single target. In this work, we constrain the adversary to select a pure strategy. Given that the defender has limited resources (e.g., she may need to protect 8 targets with 3 guards), she must design her mixed strategy to optimize against the adversary's response to maximize effectiveness.

3 Related Work

Motivated by various applications, there have been many algorithms developed to compute optimal defender strategies in Stackelberg games[6,8]. One leading family of algorithms to compute such mixed strategies are DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) [8] and its successors [6,9], which are used in the deployed ARMOR and IRIS applications. These algorithms formulate the problem as a Mixed Integer Linear Program (MILP), and compute an optimal mixed strategy for the defender assuming that the attacker responds optimally. However, in many real world domains, agents face human adversaries whose behavior may not be optimal under perfect rationality. Recent work [10] developed a new algorithm COBRA, which provided a solution for designing better defender strategies against human adversaries by accounting for their bounded rationality on computing the optimal strategy; and anchoring biases caused by limited observation conditions of the defender's strategy. COBRA outperforms DOBSS with statistical significance in experiments using human subjects, and represents the best available benchmark for how to determine defender strategies in security games against human adversaries.

This paper introduces alternative methods for computing strategies to play against human adversaries, based on two well-known theories from the behavioral literature, Prospect Theory (PT) and Quantal Response Equilibrium (QRE).

Prospect Theory is the subject of a Nobel Prize winning work. It provides a descriptive model of how humans make decisions among alternatives with risk.



(a) weighting function (b) value function

Fig. 1. PT functions from Hastie et al.

The theory describes such a decision-making process as a process of maximizing the so-called 'prospect'. Prospect is defined as $\sum_i \pi(p_i) \cdot V(C_i)$, where p_i is the probability of receiving C_i as the outcome. The weighting function $\pi(\cdot)$ describes how probability p_i is perceived by individuals.The key concepts of a weighting function are that individuals overestimate low probability and underestimate high probability [4,5], shown in Fig. 1(a). Also, $\pi(\cdot)$ is not consistent with the definition of probability in the sense that $\pi(p) + \pi(1-p) \leq 1$ in general. The value function $V(C_i)$ reflects the value of the outcome C_i. PT indicates that individuals are risk averse regarding gain but risk seeking regarding loss, implying an S-shaped value function [4,5], as shown in Fig. 1(b). A key component of Prospect Theory is the reference point. Outcomes lower than the reference point are considered as loss and higher as gain.

Quantal Response Equilibrium is an important model in behavior theory [7] and is the baseline model of many studies [12,15]. It suggests that instead of strictly maximizing utility, individuals respond stochastically in games: the chance of selecting a non-optimal strategy increases as the cost of such an error decreases. Recent work [15] shows Quantal Level-k[3] [12] to be best suited for predicting human behavior in simultaneous move games. However, the applicability of QRE and PT to security games and their comparison with COBRA remain open questions.

4 Defender Mixed-Strategy Computation

We now describe efficient computation of the optimal defender mixed strategy assuming a human adversary's response is based on either PT or QRE.

4.1 Methods for Computing PT

Best Response to Prospect Theory **(BRPT)** is a mixed integer programming formulation for the optimal leader strategy against players whose response fol-

[3] We applied QRE instead of Quantal Level-k because in Stackelberg security games the attacker observes the defender's strategy, so level-k reasoning is not applicable.

lows a PT model. Only the adversary is modeled using PT in this case, since the defender's actions are recommended by the decision aid. BRPT maximizes d, the defender's expected utility. The defender has a limited number of resources, Υ, to protect the set of targets, $t_i \in T$ for i=1..n. The defender selects a mixed strategy x that describes the probability that each target will be protected by a resource; we denote these individual probabilities by x_i. The attacker chooses a target to attack after observing x. We denote the attacker's choice using the vector of binary variables q_i, where q_i=1 if t_i is attacked and 0 otherwise.

$$\max_{x,q,a,d,z} \; d$$

$$\text{s.t.} \sum_{i=1}^{n}\sum_{k=1}^{5} x_{ik} \leq \Upsilon \tag{1}$$

$$\sum_{k=1}^{5}(x_{ik} + \bar{x}_{ik}) = 1, \forall i \tag{2}$$

$$0 \leq x_{ik}, \bar{x}_{ik} \leq c_k - c_{k-1}, \forall i, k = 1..5 \tag{3}$$

$$z_{ik} \cdot (c_k - c_{k-1}) \leq x_{ik}, \forall i, k = 1..4 \tag{4}$$

$$\bar{z}_{ik} \cdot (c_k - c_{k-1}) \leq \bar{x}_{ik}, \forall i, k = 1..4 \tag{5}$$

$$x_{i(k+1)} \leq z_{ik}, \forall i, k = 1..4 \tag{6}$$

$$\bar{x}_{i(k+1)} \leq \bar{z}_{ik}, \forall i, k = 1..4 \tag{7}$$

$$z_{ik}, \bar{z}_{ik} \in \{0, 1\}, \forall i, k = 1..4 \tag{8}$$

$$x'_i = \sum_{k=1}^{5} b_k x_{ik},\ \bar{x}'_i = \sum_{k=1}^{5} b_k \bar{x}_{ik}, \forall i \tag{9}$$

$$\sum_{i=1}^{n} q_i = 1,\ q_i \in \{0, 1\}, \forall i \tag{10}$$

$$0 \leq a - (x'_i(P_i^a)' + \bar{x}'_i(R_i^a)') \leq M(1 - q_i), \forall i \tag{11}$$

$$M(1 - q_i) + \sum_{k=1}^{5}(x_{ik}R_i^d + \bar{x}_{ik}P_i^d) \geq d, \forall i \tag{12}$$

The defender optimization problem is given in Equations (1)-(12). In security games, the payoffs depend only on whether or not the attack was successful, so given a target t_i, the defender (resp., adversary) receives reward R_i^d (penalty P_i^a) if the adversary attacks the target and it is covered by the defender; otherwise, the defender (adversary) receives penalty P_i^d (reward R_i^a).

PT comes into the algorithm by adjusting the weighting and value functions as described above. The benefit (prospect) perceived by the adversary for attacking target t_i if the defender plays the mixed strategy x is given by $\pi(x_i)V(P_i^a) + \pi(1 - x_i)V(R_i^a)$. Let $(P_i^a)' = V(P_i^a)$ and $(R_i^a)' = V(R_i^a)$ denote the adversary's value of penalty P_i^a and reward R_i^a. We use a piecewise linear function $\tilde{\pi}(\cdot)$ to approximate the non-linear weighting function $\pi(\cdot)$ and empiri-

cally set 5 segments[4] for $\tilde{\pi}(\cdot)$. This function is defined by $\{c_k|c_0 = 0, c_5 = 1, c_k < c_{k+1}, k = 0, ..., 5\}$ that represent the endpoints of the linear segments and $\{b_k|k = 1, \ldots, 5\}$ that represent the slope of each linear segment. Thus, each of the defender's x_i=$\sum_{k=1}^{5} x_{ik}$; the follower will perceive this x_i as x'_i=$\pi(x_i)$=$\sum_{k=1}^{5} b_k \cdot x_{ik}$ as discussed below.

In order to represent the piecewise linear function, we break x_i (and $1 - x_i$) into five segments, denoted by variable x_{ik} (and $\bar{x}_{ik}$). We can enforce that such breakup of x_i (and $1 - x_i$) is correct if segment x_{ik} (and $\bar{x}_{ik}$) is positive only if the previous segment is used completely, for which we need the auxiliary integer variable z_{ik} (and $\bar{z}_{ik}$). This is enforced by Equations (3)$\sim$(8). Equation (9) defines x'_i and $\bar{x}'_i$ as the value of the piecewise linear approximation of x_i and $1 - x_i$: x'_i=$\tilde{\pi}(x_i)$ and $\bar{x}'_i$=$\tilde{\pi}(1 - x_i)$. Equations (10) and (11) define the optimal adversary's pure strategy. In particular, Equation (11) enforces that q_i=1 for the action that achieves maximal prospect for the adversary. Equation (12) enforces that d is the defender's expected utility on the target that is attacked by the adversary (q_i=1).

Robust-PT (RPT) modifies the base BRPT method to account for some uncertainty about the adversaries choice, caused (for example) by imprecise computations [11]. Similar to COBRA, RPT assumes that the adversary may choose any strategy within ϵ of the best choice, defined here by the prospect of each action. It optimizes the worst-case outcome for the defender among the set of strategies that have prospect for the attacker within ϵ of the optimal prospect. We modify the BRPT optimization problem as follows: the first 11 Equations are equivalent to those in BRPT; in Equation (13), the binary variable h_i indicates all the $\epsilon-$optimal strategies for the adversary; Equation (16) enforces that d is the minimum expected utility of the defender against the $\epsilon-$optimal strategies of the adversary.

$$\max_{x,h,q,a,d,z} d$$

s.t. Equations (1)$\sim$(11)

$$\sum_{i=1}^{n} h_i \geq 1 \tag{13}$$

$$h_i \in \{0, 1\},\ q_i \leq h_i, \forall i \tag{14}$$

$$\epsilon(1 - h_i) \leq a - (x'_i(P_i^a)' + \bar{x}'_i(R_i^a)') \leq M(1 - h_i), \forall i \tag{15}$$

$$M(1 - h_i) + \sum_{k=1}^{5}(x_{ik}R_i^d + \bar{x}_{ik}P_i^d) \geq d, \forall i \tag{16}$$

Runtime: We choose AMPL (http://www.ampl.com/) to solve the MILP with CPLEX as the solver. Both BRPT and RPT take less than 1 second for up to 10 targets.

[4] This piecewise linear representation of $\pi(\cdot)$ can achieve a small approximation error: $\sup_{z\in[0,1]} \|\pi(z) - \tilde{\pi}(z)\| \leq 0.03$.

4.2 Methods for Computing QRE

In applying the QRE model to our domain, we only add noise to the response function for the adversary, so the defender computes an optimal strategy assuming the attacker response with a noisy best-response. The parameter λ represents the amount of noise in the attacker's response. Given λ and the defender's mixed-strategy x, the adversaries' quantal response q_i (i.e. probability of i) can be written as

$$q_i = \frac{e^{\lambda U_i^a(x)}}{\sum_{j=1}^{n} e^{\lambda U_j^a(x)}} \tag{17}$$

where, $U_i^a(x) = x_i P_i^a + (1-x_i) R_i^a$ is the adversary's expected utility for attacking t_i and x is the defender's strategy.

$$q_i = \frac{e^{\lambda R_i^a} e^{-\lambda (R_i^a - P_i^a) x_i}}{\sum_{j=1}^{n} e^{\lambda R_j^a} e^{-\lambda (R_j^a - P_j^a) x_j}} \tag{18}$$

The goal is to maximize the defender's expected utility given q_i, i.e. $\sum_{i=1}^{n} q_i (x_i R_i^d + (1 - x_i) P_i^d)$. Combined with Equation (18), the problem of finding the optimal mixed strategy for the defender can be formulated as

$$\begin{aligned} \max_{x} \quad & \frac{\sum_{i=1}^{n} e^{\lambda R_i^a} e^{-\lambda (R_i^a - P_i^a) x_i} ((R_i^d - P_i^d) x_i + P_i^d)}{\sum_{j=1}^{n} e^{\lambda R_j^a} e^{-\lambda (R_j^a - P_j^a) x_j}} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq \Upsilon \\ & 0 \leq x_i \leq 1, \quad \forall i, j \end{aligned} \tag{19}$$

Given that the objective function in Equation (19) is non-linear and non-convex in its most general form, finding the global optimum is extremely difficult. Therefore, we focus on methods to find local optima. To compute an approximately optimal QRE strategy efficiently, we develop the Best Response to Quantal Response (**BRQR**) heuristic described in Algorithm 1. We first take the negative of Equation (19), converting the maximization problem to a minimization problem. In each iteration, we find the local minimum[5] using a gradient descent technique from the given starting point. If there are multiple local minima, by randomly setting the starting point in each iteration, the algorithm will reach different local minima with a non-zero probability. By increasing the iteration number, $IterN$, the probability of reaching the global minimum increases.

Parameter Estimation: The parameter λ in the QRE model represents the amount of noise in the best-response function. One extreme case is λ=0, when play becomes uniformly random. The other extreme case is λ=∞, when the quantal response is identical to the best response. λ is sensitive to game payoff structure, so tuning λ is a crucial step in applying the QRE model. We employed Maximum Likelihood Estimation (MLE) to fit λ using data from [10].

[5] We use *fmincon* function in Matlab to find the local minimum.

Algorithm 1 BRQR

```
opt_g ← −∞;                                          ▷ Initialize the global optimum
for i ← 1, ..., IterN do
    x_0 ← randomly generate a feasible starting point
    (opt_l, x*) ← FindLocalMinimum(x_0)
    if opt_g > opt_l then
        opt_g ← opt_l, x_opt ← x*
    end if
end for
return opt_g, x_opt
```

Given the defender's mixed strategy x and N samples of the players' choices, the logit likelihood of λ is

$$\log L(\lambda \mid x) = \sum_{j=1}^{N} \log q_{\tau(j)}(\lambda)$$

where $\tau(j)$ denotes the target attacked by the player in sample j. Let N_i be the number of subjects attacking target i. Then, we have $\log L(\lambda \mid x)=\sum_{i=1}^{n} N_i \log q_i(\lambda)$. Combining with Equation (17),

$$\log L(\lambda \mid x) = \lambda \sum_{i=1}^{n} N_i U_i^a(x) - N \cdot \log(\sum_{i=1}^{n} e^{\lambda U_i^a(x)})$$

$\log L(\lambda \mid x)$ is a concave function[6]. Therefore, $\log L(\lambda \mid x)$ only has one local maximum. The MLE of λ is 0.76 for the data used from [10].

Runtime: We implement BRQR in Matlab. With 10 targets and $IterN$=300, the runtime of BRQR is less than 1 minute. In comparison, with only 4 targets, LINGO12 (http://www.lindo.com/) cannot compute the global optimum of Equation (19) within one hour.

5 Payoff Structure Classification

One important property of payoff structures we want to examine is their influence on model performance. We certainly cannot test over all possible payoff structures, so the challenges are: (i) the payoff structures we select should be representative of the payoff structure space; (ii) the strategies generated from different algorithms should be sufficiently separated. As we will discuss later, the payoff structures used in [10] do not address these challenges.

[6] The second order derivative of $\log L(\lambda \mid x)$ is

$$\frac{d^2 \log L}{d\lambda^2} = \frac{\sum_{i<j} -(U_i^a(x) - U_j^a(x))^2 e^{\lambda(U_i^a(x)+U_j^a(x))}}{(\sum_i e^{\lambda U_i^a(x)})^2} < 0$$

We address the first criterion by randomly sampling 1000 payoff structures, each with 8 targets. R_i^a and R_i^d are integers drawn from $Z^+[1,10]$; P_i^a and P_i^d are integers drawn from $Z^-[-10,-1]$. This scale is similar to the payoff structures used in [10]. We then clustered the 1000 payoff structures into four clusters using k-means clustering based on eight features, which are defined in Table 1. Intuitively, features 1 and 2 describe how 'good' the game is for the adversary,

Table 1. A-priori defined features

Feature 1	Feature 2	Feature 3	Feature 4
mean($\lvert\frac{R_i^a}{P_i^a}\rvert$)	std($\lvert\frac{R_i^a}{P_i^a}\rvert$)	mean($\lvert\frac{R_i^d}{P_i^d}\rvert$)	std($\lvert\frac{R_i^d}{P_i^d}\rvert$)
Feature 5	Feature 6	Feature 7	Feature 8
mean($\lvert\frac{R_i^a}{P_i^d}\rvert$)	std($\lvert\frac{R_i^a}{P_i^d}\rvert$)	mean($\lvert\frac{R_i^d}{P_i^a}\rvert$)	std($\lvert\frac{R_i^d}{P_i^a}\rvert$)

features 3 and 4 describe how 'good' the game is for the defender, and features 5∼8 reflect the level of 'conflict' between the two players in the sense that they measure the ratio of one player's gain over the other player's loss. In Fig. 2,

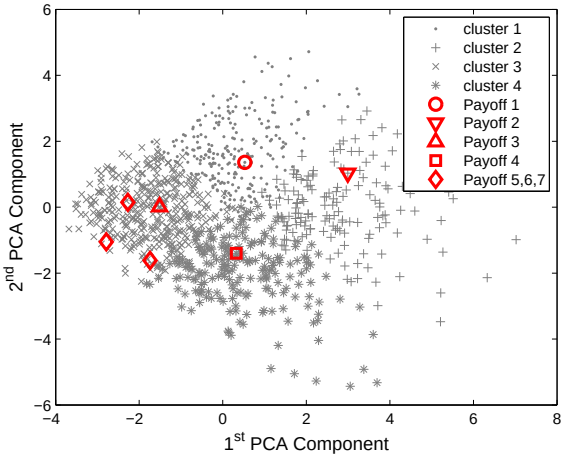


Fig. 2. Payoff Structure Clusters (color)

all 1000 payoff structures are projected onto the first two Principal Component Analysis (PCA) dimensions for visualization. We select one payoff structure from each cluster, following the criteria below to obtain sufficiently different strategies for the different candidate algorithms:

– We define the distance between two mixed strategies, x^k and x^l, using the Kullback-Leibler divergence:

$$D(x^k, x^l) = D_{KL}(x^k|x^l) + D_{KL}(x^l|x^k)$$

where, $D_{KL}(x^k|x^l) = \sum_{i=1}^{n} x_i^k \log(x_i^k/x_i^l)$.
- For each payoff structure, $D(x^k, x^l)$ is measured for every pair of strategies. With five strategies (discussed later), we have 10 such measurements.
- We remove payoff structures that have a mean or minimum of these 10 quantities below a given threshold. This gives us a subset of about 250 payoff structures in each cluster. We then select one payoff structure closest to the cluster center from the subset of each cluster .

The four payoff structures (payoffs 1-4) we selected from each cluster are marked in Fig. 2, as are the three (payoffs 5-7) used in [10]. Fig. 2 shows that payoffs 5-7 all belong to cluster 3. Furthermore, Table 2 reports the strategy distances

Table 2. Strategy Distance

Payoff Structure	1	2	3	4	5	6	7
mean D_{KL}	0.83	1.19	0.64	0.88	0.32	0.15	0.12
min D_{KL}	0.26	0.25	0.21	0.25	0.07	0.02	0.04

in all seven payoff structures. The strategies are not as well separated in payoffs 5-7 as they are in payoffs 1-4. As we discuss in Section. 6.2, the performance of different strategies is quite similar in payoffs 5-7.

6 Experiments

We conducted empirical tests with human subjects playing an online game to evaluate the performances of leader strategies generated by five candidate algorithms. We based our model on the LAX airport, which has eight terminals that can be targeted in an attack [9]. Subjects play the role of followers and are able to observe the leader's mixed strategy (i.e., randomized allocation of security resources).

6.1 Experimental Setup

Fig. 3 shows the interface of the web-based game we developed to present subject with choice problems. Players were introduced to the game through a series of explanatory screens describing how the game is played. In each game instance a subject was asked to choose one of the eight gates to open (attack). They knew that guards were protecting three of the eight gates, but not which ones. Subjects were rewarded based on the reward/penalty shown for each gate and the probability that a guard was behind the gate (i.e., the exact randomized strategy of the defender). To motivate the subjects they would earn or lose money based on whether or not they succeed in attacking a gate; if the subject opened a gate not protected by the guards, they won; otherwise, they lost. Subjects start with an endowment of $8 and each point won or lost in a game instance was worth $0.1. On average, subjects earned about $14.1 in cash.

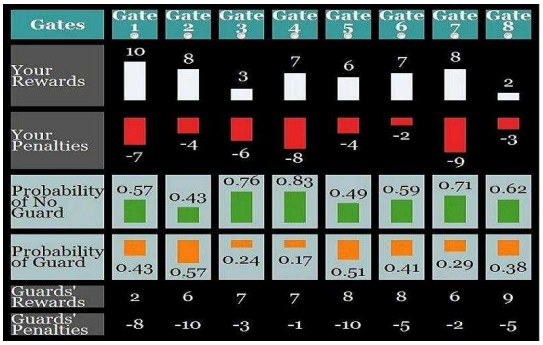


Fig. 3. Game Interface

We tested the seven different payoff structures from Fig. 2 (four new, three from [10]). The seven payoff structures are displayed in Table 4. For each payoff structure we tested the mixed strategies generated by five algorithms: BRPT, RPT, BRQR, COBRA and DOBSS, which are reported in Table 5 and Table 6. There were a total of 35 payoff structure/strategy combinations and each subject played all 35 combinations. In order to mitigate the order effect on subject responses, a total of 35 different orderings of the 35 combinations were generated using Latin Square design. Every ordering contained each of the 35 combinations exactly once, and each combination appeared exactly once in each of the 35 positions across all 35 orderings. The order played by each subject was drawn uniformly randomly from the 35 possible orderings. To further mitigate learning, no feedback on success or failure was given to the subjects until the end of the experiment. A total of 40 human subjects played the game.

We could explore only a limited number of parameters for each algorithm, which were selected following the best available information in the literature. The parameter settings for each algorithm are reported in Table 3. DOBSS has

Table 3. Model Parameter

Payoff Structure	1	2	3	4	5	6	7
RPT-ϵ	2.4	3.0	2.1	2.75	1.9	1.5	1.5
COBRA-α	0.15	0.15	0.15	0.15	0.37	0	0.25
COBRA-ϵ	2.5	2.9	2.0	2.75	2.5	2.5	2.5

no parameters. The values of PT parameters are typical values reported in the literature [4]. We set ϵ in RPT following two rules: (i) No more than half of targets are in the $\epsilon-$optimal set; (ii) $\epsilon \leq 0.3R^a_{max}$, where R^a_{max} is the maximum potential reward for the adversary. The size of the $\epsilon-$optimal set increases as the value of ϵ increases. When ϵ is sufficiently large, the defender's strategy becomes maximin, since she believes that the adversary may attack any target. The second rule limits the imprecision in the attacker's choice. We empirically set the limit to $0.3R^a_{max}$. For BRQR, we set λ using MLE with data reported

in [10] (see Section 4.2). For payoffs 1∼4, we set the parameters for COBRA following the advices given by [10] as close as possible. In particular, the values we set for α meet the entropy heuristic discussed in that work. For payoffs 5∼7, we use the same parameter settings as in their work.

6.2 Experiment Result

Quality Comparison: We used the average expected defender's utility to evaluate the performances of the strategies. Let $U_i^d(x) = x_i R_i^d + (1 - x_i)P_i^d$ be the defender's expected utility for target t_i if she plays mixed strategy x and the subject selects target t_i; N_i be the number of subjects that chose target t_i. Then, the average expected defender's utility can is calculated as

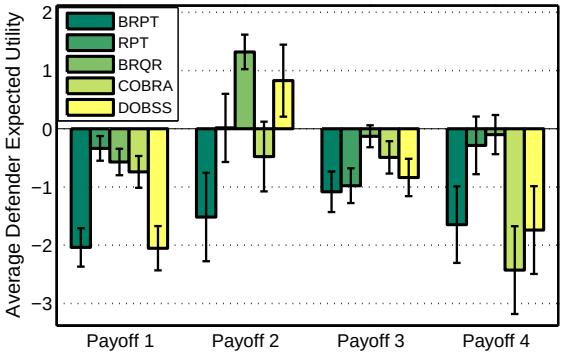
$$\bar{U}_{exp}^d(x) = \frac{1}{N}\sum_{i=1}^{n} N_i U_i^d(x)$$

Fig. 4 displays $\bar{U}_{exp}^d(x)$ for the different strategies in each payoff structure. The performance of the strategies is closer in payoffs 5∼7 than in payoffs 1∼4. The main reason is that strategies are not very different in payoffs 5∼7 (see Table 2). We evaluate the statistical significance of our results using the bootstrap-t method [14]. The comparison is summarized below:

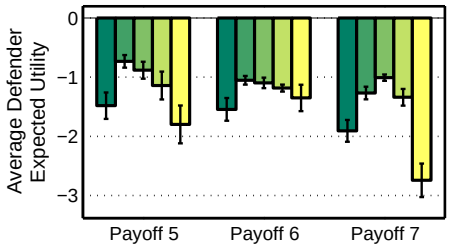
- BRQR outperforms COBRA in all seven payoff structures. The result is statistically significant in three cases ($p<0.005$) and borderline ($p=0.05$) in payoff 3 ($p<0.06$). BRQR also outperforms DOBSS in all cases, with statistical significance in five of them ($p<0.02$).
- RPT outperforms COBRA except in payoff 3. The difference is statistically significant in payoff 4 ($p<0.005$). In payoff 3, COBRA outperforms RPT ($p>0.07$). Meanwhile, RPT outperforms DOBSS in five payoff structures, with statistical significance in four of them ($p<0.05$). In the other two cases, DOBSS has better performance ($p>0.08$).
- BRQR outperforms RPT in three payoff structures with statistical significance ($p<0.005$). They have very similar performance in the other four cases.
- BRPT is outperformed by BRQR in all cases with statistical significance ($p<0.03$). It is also outperformed by RPT in all cases, with statistical significance in five of them ($p<0.02$) and one borderline ($p<0.06$). BRPT's failure to perform better (and even worse than COBRA) is a surprising outcome.

Overall, BRQR performs best, RPT outperforms COBRA in six of the seven cases, and BRPT and DOBSS perform the worst.

Key Observations: BRPT and DOBSS are not robust against an adversary that deviates from the optimal strategy. BRQR, RPT and COBRA all try to be robust against such deviations. BRQR considers some (possibly very small) probability of adversary attacking any target. In contrast, COBRA and RPT separate the targets into two groups, the ϵ-optimal set and the non-ϵ-optimal set, using a hard threshold. They then try to maximize the worst case for the



(a) New Payoffs



(b) Payoffs from Pita et al.

Fig. 4. Average Expected Utility of Defender

defender assuming the response will be in the ϵ-optimal set, but assign less resources to other targets. When the non-ϵ-optimal targets have high defender penalties, COBRA and RPT become vulnerable, especially in the following two cases:

- 'Unattractive' targets are those with small reward but large penalty for the adversary. COBRA and RPT consider such targets as non-ϵ-optimal and assign significantly less resources than BRQR on them. However, some subjects would still select such targets and caused severe damage to COBRA and RPT (e.g. about 30% subjects selected door 5 in payoff 4 against COBRA strategy, as shown in Fig. 5(d)).
- 'High-risk' targets are those with large reward and large penalty for the adversary. RPT considers such targets as non-ϵ-optimal and assigns far less resources than other algorithms. This is caused by the assumptions made by PT that people care more about loss than gain and that they overestimate small probabilities. However, experiments show RPT gets hurt significantly on such targets (e.g. more than 15% subjects select door 1 in payoff 2, as shown in Fig. 5(b)).

7 Conclusions

There is a significant interest in game-theoretic techniques to solve security problems. However, current algorithms make perfect rationality assumption of the adversaries, which is problematic in many real security domains when agents face human adversaries. New methods need to be developed to compute defender strategy against real human adversaries. This paper successfully integrates two important human behavior theories, PT and QRE, into building more realistic decision-support tool. To that end, the main contributions of this paper are, (i) Developing efficient new algorithms based on PT and QRE models of human behavior; (ii) Conducting the most comprehensive experiments to date with human subjects for security games (40 subjects, 5 strategies, 7 game structures); (iii) Designing techniques for generating representative payoff structures for behavioral experiments in generic classes of games. By providing new algorithms that outperform the leading competitor, this paper has advanced the state-of-the-art.

References

1. C. F. Camerer, T. Ho, and J. Chongn. A congnitive hierarchy model of games. *QJE*, 119(3):861–898, 2004.
2. S. Ficici and A. Pfeffer. Simultaneously modeling humans' preferences and their beliefs about others' preferences. *AAMAS*, 2008.
3. Y. Gal and A. Pfeffer. Modeling reciprocal behavior in human bilateral negotiation. *AAMAS*, 2007.
4. R. Hastie and R. M. Dawes. *Rational Choice in an Uncertain World: the Psychology of Judgement and Decision Making.* Sage Publications, Thounds Oaks, 2001.
5. D. Kahneman and A. Tvesky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–292, 1979.
6. C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordonez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. *In AAMAS*, 2009.
7. R. D. McKelvey and T. R. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behavior*, 2:6–38, 1995.
8. P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. *In AAMAS*, 2008.
9. J. Pita, M. Jain, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. *In AAMAS*, 2008.
10. J. Pita, M. Jain, F. Ordonez, M. Tambe, and S. Kraus. Solving stackelberg games in the real-world: Addressing bounded rationality and limited observations in human preference models. *Artificial Intelligence Journal*, 174(15):1142–1171, 2010.
11. H. Simon. Rational choice and the structure of the environment. *Psychological Review*, 63(2):129–138, 1956.
12. D. O. Stahl and P. W. Wilson. Experimental evidence on players' models of other players. *JEBO*, 25(3):309–327, 1994.

13. J. Tsai, S. Rathi, C. Kiekintveld, F. Ordonez, and M. Tambe. Iris - a tool for strategic security allocation in transportation networks. *In AAMAS*, 2009.
14. R. R. Wilcox. *Applying contemporary statistical techniques*. Academic Press, 2003.
15. J. R. Wright and K. Leyton-Brown. Beyond equilibrium: Predicting human behavior in normal-form games. *In AAAI*, 2010.

A Payoff Structure Information

The four payoff structures selected from the four clustering groups are displayed in Table 4.

Table 4. Payoff Structure

(a) Payoff Structure 1

Target	1	2	3	4	5	6	7	8
defender reward	2	6	7	7	8	8	6	9
defender penalty	-8	-10	-3	-1	-10	-5	-2	-5
subject reward	10	8	3	7	6	7	8	2
subject penlaty	-7	-4	-6	-8	-4	-2	-9	-3

(b) Payoff Structure 2

Target	1	2	3	4	5	6	7	8
defender reward	3	8	9	9	7	7	4	1
defender penalty	-10	-2	-5	-1	-7	-6	-2	-1
subject reward	9	8	2	9	10	1	10	1
subject penlaty	-10	-1	-10	-8	-4	-10	-5	-3

(c) Payoff Structure 3

Target	1	2	3	4	5	6	7	8
defender reward	5	3	8	3	3	4	3	6
defender penalty	-2	-5	-4	-6	-3	-10	-7	-2
subject reward	8	6	1	3	1	7	3	5
subject penlaty	-6	-9	-3	-7	-7	-2	-5	-2

(d) Payoff Structure 4

Target	1	2	3	4	5	6	7	8
defender reward	5	9	10	2	10	4	8	8
defender penalty	-10	-4	-9	-3	-10	-10	-2	-5
subject reward	3	7	3	9	2	9	7	8
subject penlaty	-4	-8	-5	-8	-9	-4	-1	-6

(e) Payoff Structure 5

Target	1	2	3	4	5	6	7	8
defender reward	1	4	2	3	4	1	5	2
defender penalty	-5	-8	-1	-6	-5	-1	-7	-7
subject reward	1	9	5	6	7	1	10	3
subject penlaty	-2	-4	-3	-3	-3	-2	-4	-3

(f) Payoff Structure 6

Target	1	2	3	4	5	6	7	8
defender reward	4	3	1	5	1	2	5	2
defender penalty	-8	-10	-1	-8	-1	-3	-11	-5
subject reward	8	5	3	10	1	3	9	4
subject penlaty	-3	-2	-3	-2	-3	-3	-2	-3

(g) Payoff Structure 7

Target	1	2	3	4	5	6	7	8
defender reward	4	3	1	5	1	2	5	2
defender penalty	-8	-5	-1	-10	-5	-3	-9	-6
subject reward	8	5	2	10	1	3	9	4
subject penlaty	-3	-3	-3	-3	-3	-3	-3	-3

B Defender Mixed-Strategy

The defender's mixed-strategy from each algorithm in each payoff structures are displayed in Table 5 and Table 6.

Table 5. Defender's Mixed-strategy

(a) Payoff Structure 1

Target	1	2	3	4	5	6	7	8
BRPT	0.39	0.51	0.17	0.26	0.43	0.70	0.26	0.28
RPT	0.43	0.57	0.24	0.17	0.51	0.41	0.29	0.38
BRQR	0.57	0.58	0.18	0.21	0.51	0.47	0.30	0.18
COBRA	0.57	0.62	0.18	0.22	0.51	0.44	0.34	0.11
DOBSS	0.49	0.53	0.15	0.36	0.44	0.59	0.37	0.07

(b) Payoff Structure 2

Target	1	2	3	4	5	6	7	8
BRPT	0.28	0.93	0.07	0.34	0.59	0.05	0.52	0.23
RPT	0.32	0.54	0.10	0.39	0.65	0.07	0.57	0.37
BRQR	0.54	0.52	0.21	0.36	0.64	0.16	0.58	0.00
COBRA	0.48	0.53	0.09	0.43	0.74	0.00	0.70	0.02
DOBSS	0.42	0.78	0.08	0.47	0.64	0.00	0.60	0.00

(c) Payoff Structure 3

Target	1	2	3	4	5	6	7	8
BRPT	0.42	0.24	0.29	0.19	0.09	0.78	0.28	0.72
RPT	0.46	0.27	0.38	0.23	0.12	0.80	0.34	0.40
BRQR	0.36	0.43	0.20	0.36	0.13	0.72	0.43	0.37
COBRA	0.48	0.42	0.16	0.29	0.07	0.81	0.36	0.42
DOBSS	0.53	0.37	0.12	0.25	0.06	0.72	0.31	0.64

(d) Payoff Structure 4

Target	1	2	3	4	5	6	7	8
BRPT	0.28	0.27	0.22	0.33	0.08	0.54	0.90	0.38
RPT	0.37	0.32	0.30	0.37	0.10	0.61	0.49	0.44
BRQR	0.35	0.33	0.30	0.44	0.20	0.62	0.36	0.42
COBRA	0.24	0.42	0.21	0.50	0.04	0.66	0.39	0.53
DOBSS	0.22	0.37	0.19	0.44	0.05	0.58	0.69	0.47

C Histogram of Subjects' Choices

The histograms of subjects' choice in each game instance are displayed in Fig. 5 and Fig. 6.

Table 6. Defender's Mixed-strategy

(a) Payoff Structure 5

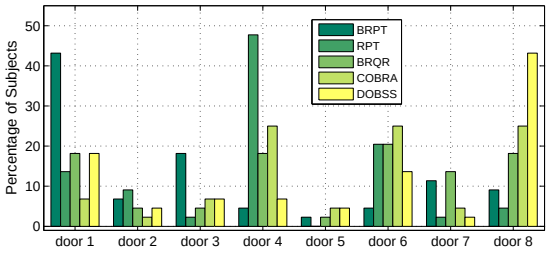
Target	1	2	3	4	5	6	7	8
BRPT	0.16	0.49	0.41	0.46	0.51	0.16	0.52	0.28
RPT	0.23	0.52	0.17	0.50	0.50	0.23	0.54	0.32
BRQR	0.12	0.61	0.16	0.55	0.52	0.00	0.57	0.46
COBRA	0.00	0.64	0.23	0.63	0.52	0.00	0.55	0.40
DOBSS	0.00	0.59	0.45	0.51	0.56	0.00	0.62	0.27

(b) Payoff Structure 6

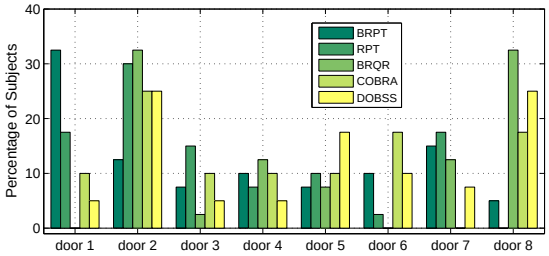
Target	1	2	3	4	5	6	7	8
BRPT	0.49	0.46	0.21	0.67	0.05	0.21	0.64	0.29
RPT	0.54	0.53	0.07	0.55	0.07	0.27	0.63	0.35
BRQR	0.58	0.59	0.00	0.60	0.00	0.19	0.66	0.38
COBRA	0.58	0.55	0.00	0.53	0.00	0.31	0.62	0.41
DOBSS	0.56	0.45	0.19	0.68	0.00	0.19	0.65	0.30

(c) Payoff Structure 7

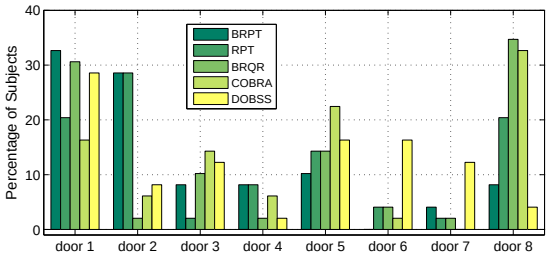
Target	1	2	3	4	5	6	7	8
BRPT	0.54	0.41	0.19	0.60	0.09	0.27	0.57	0.35
RPT	0.56	0.43	0.03	0.60	0.12	0.31	0.58	0.38
BRQR	0.59	0.44	0.00	0.63	0.08	0.22	0.60	0.45
COBRA	0.57	0.48	0.00	0.59	0.00	0.33	0.56	0.47
DOBSS	0.59	0.44	0.10	0.65	0.00	0.25	0.62	0.36



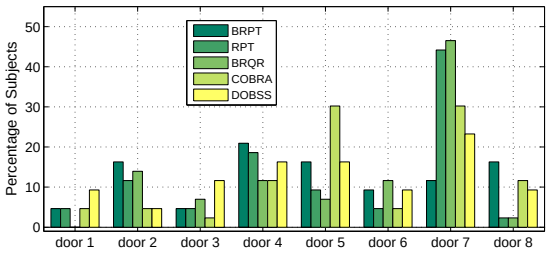
(a) Payoff Structure 1



(b) Payoff Structure 2

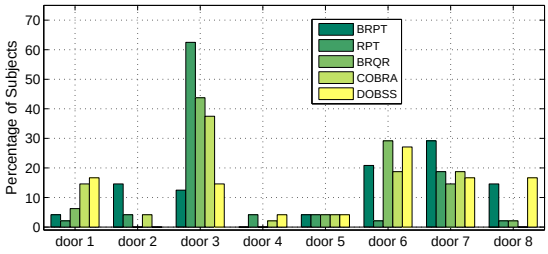


(c) Payoff Structure 3

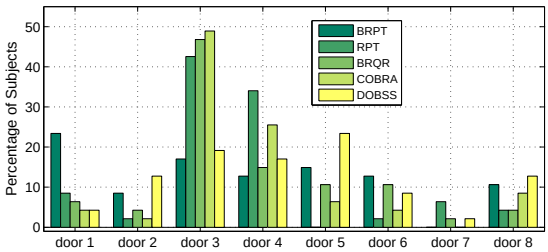


(d) Payoff Structure 4

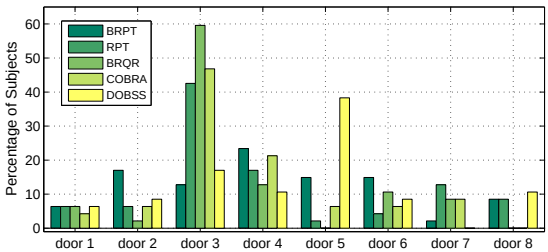
Fig. 5. Histogram of Subjects' Choices



(a) Payoff Structure 5



(b) Payoff Structure 6



(c) Payoff Structure 7

Fig. 6. Histogram of Subjects' Choices

Towards Realistic Decentralized Modeling for Use in a Real-World Personal Assistant Agent Scenario

Christopher Amato, Nathan Schurr, and Paul Picciano

Aptima, Inc.
12 Gill Street, Suite 1400
Woburn, MA 01801
{camato,nschurr,ppicciano}@aptima.com

Abstract. Many approaches have been introduced for representing and solving multiagent coordination problems. Unfortunately, these methods make assumptions that limit their usefulness when combined with human operators and real-life hardware and software. In this paper, we discuss the problem of using agents in conjunction with human operators to improve coordination as well as possible models that could be used in these problems. Our approach – Space Collaboration via an Agent Network (SCAN) – enables proxy agents to represent each of the stakeholder agencies in a space setting and shows how the SCAN agent network could facilitate collaboration by identifying opportunities and methods of collaboration. We discuss this approach as well as the challenges in extending models to 1) take advantage of human input, 2) deal with the limited and uncertain information that will be present and 3) combat the scalability issues in solution methods for a large number of decentralized agents. As a step toward dealing with limited information, we propose the *shared MDP* model, which allows private and shared information to be expressed separately.

1 Introduction

In today's world, people are increasingly connected to many other people and many sources of information, providing more opportunities than ever for tasks such as working more efficiently with others and discovering helpful information. Unfortunately, the vast scale of information can be overwhelming. As computers become more powerful and pervasive, software in the form of personal assistant agents can provide support in many situations.

For example, with the increased deployment and use of space assets, a number of interesting and challenging problems have come to the fore. The persistent nature of space surveillance (i.e., 24/7 operations), the mass of data, the varied data sources, and the heterogeneous needs of consumers and producers throughout the community all point to a pressing need for an enhanced Space Situational Awareness (SSA) picture, one that can only be achieved by a coordinated, collaborative approach to the collection, dissemination and sharing of information.

For the foreseeable future, the challenges in the U.S. National Space Policy will demand human-in-the-loop participation. This is particularly relevant given the amount and importance of data and knowledge not held in any database, or streaming in bits

through the ionosphere, but that resides in the minds of individuals or the institutional knowledge of a team. The criticality of an enhanced situational awareness and its subsequent deployment suggests that we integrate these methods within the existing workflow, obviating the need to add additional tools.

The solution to this problem requires a coherent, integrated approach, viewing the space domain as a socio-technical system, within which the human plays an integral part. Several diverse agencies are stakeholders and produce and/or consume information related to space. Each agency has its own organizational structure and protocols, so the solution must be versatile enough to allow for effective inter-agency collaboration while still maintaining the standard practices of each. Further, the human operator network alone is not sufficient to create adequate SSA. There is an enormous amount of data recorded from sensors on satellites, ground stations and other periodic collectors. The human operator is overloaded by the transmission, capture, cataloging, and sense making of these data. The optimal solution for enhanced SSA is a synergistic, collaborative effort between networks of humans and automated agents.

The Space Collaboration via an Agent Network (SCAN) approach enables proxy agents to represent each of the stakeholder agencies and facilitate collaboration by identifying opportunities and methods for collaboration. This allows humans to have access to large amounts of data through their agents, while personalizing them given the users specific preferences and capabilities. These agents can then communicate with each other with varying levels of input from the user for tasks ranging from retrieving information, securing the services of another agency or team formation for complex tasks.

Many models have been developed for solving decentralized problems requiring cooperation. For SCAN, we assume a sequential problem in which a series of decisions need to be made, each affecting the next. Limiting these models to those based on decision theory, a large number of models can still be used [2, 4, 5, 10, 14]. Nevertheless, each of these models makes assumptions that cause it to be inappropriate for a domain such as SCAN. These assumptions include: a full model of the problem with the value other agents derive from all actions or the results of these actions, perfect knowledge of the current state of the system (or a common estimate), a shared objective function, a centralized solution method. An ideal model for our domain would relax these assumptions to permit only local knowledge and solution methods, self-interested agents, model uncertainty and independence between groups of agents.

The remainder of this paper is organized as follows. We first discuss the SCAN domain in more detail as well as the progress made to date. We also discuss several models that could be used to represent this problem and their shortcomings. Finally, we discuss key assumptions that we are interested in relaxing in these approaches and the challenges involved in doing so. The goal in this project is to produce a system of personal assistant agents with accompanying solution methods that maintain tractability and high solution quality.

2 Overview of SCAN approach

The approach used in SCAN represents each of the stakeholder agencies for space with proxy agents and facilitates collaboration by identifying opportunities and methods for

collaboration. It demonstrates the suitability of our human-agent solution to address the three critical aspects above, that is: 1) Knowledge and mental models of human operators, 2) Collaboration methods and barriers to collaboration and 3) Integrating solutions that conform to preferred workflows.

A variety of factors shape the opportunities for collaboration and the constraints on collaboration. The SCAN Agent architecture incorporates these components into the SCAN Proxy Agent Architecture. The need to collaborate varies based on an interaction of the demands of the mission, the architecture of the team and its resources, and the distribution of expertise within the team. Collaboration becomes necessary when mission-critical expertise is distributed among multiple people, and resources and responsibilities are likewise divided between people.

A team's ability to collaborate depends on a number of factors, including available technology, team members collaboration skills, and team composition. The factors affecting the ability to collaborate directly affect the products of collaboration (e.g., assessments and plans), as well as the apparent coordination of the team as a whole. Collaborative critical thinking is intended to provide active support to teams above and beyond the three factors affecting a team's ability to collaborate. Figure 1 shows a top-level diagram of our proposed solution: Space Collaboration via an Agent Network (SCAN). It shows proxy agents, each of which represents a stakeholder for space. We implemented an initial version of this infrastructure by developing agent interactions to carry out the use case(s), and tested the resultant model by generating results to show dynamic constraints, changing parameters, and making modifications to the use cases(s) that show the benefit of agent use. This solution will fit integrally into the current workflow, without the need for additional tools for users to learn, and will make privacy issues explicit.

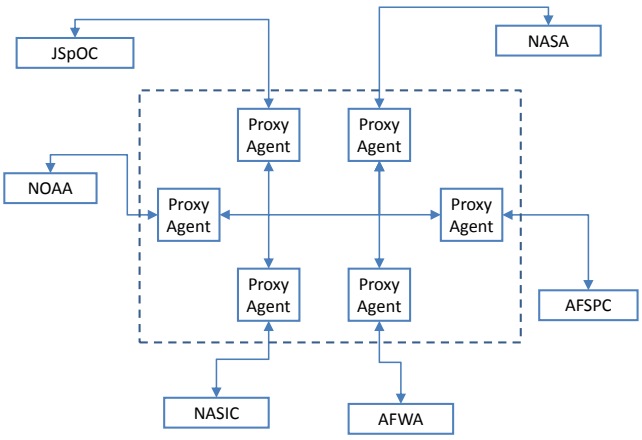


Fig. 1. Proxy agents being used to facilitate collaboration between heterogeneous stakeholders.

Note that our approach to SCAN is to have the human remain in control and drive the collaboration. The proxy agent is designed to facilitate the human operators and point them in the right direction to improve interagency collaboration. Human operators will not be surprised or superseded by the SCAN proxy agent's actions.

3 The SCAN framework

In conceiving an agent-based architecture, we identified the following benefits to individual agencies of agent-based collaboration: 1) Fusion/Integration of agency plans, 2) Locating potential collaboration opportunities, 3) Forewarning: An agency may anticipate future events based on an alert from another agency, 4) Increased Quality/Accuracy of data, 5) Facilitating levels of automated collaboration, 6) Detecting collaboration failures, 7) Adjusting to realtime changes in policy/capabilities, 8) Learning preferences of agencies, 9) Persuading agencies to cooperate through incentives. We selected areas 1), 2), and 7) as areas for further study.

3.1 Use case

In order to study the above, we constructed a working Use Case. The requirements for the Use Case were: Scalability: The initial Use Case should describe a simple problem that can easily scale to more complex problems. This allows us to demonstrate our solution initially in an easily controllable environment, but also set the stage for our solution to handle a larger set of more complex problems. Generalizability: The Use Case should be representative of a general class of collaboration problems, so that the solution will be applicable to the needs of other agencies as well as those mentioned in the Use Case. The Use Case was based on interviews with space weather analysts from the Air Force Weather Agency (AFWA), and their information needs as primary producers of information on environmental effects to agencies such as the National Air and Space Intelligence Center (NASIC), the Joint Space Operations Center (JSpOC), and the Air Operations Center (AOC)

We identified two important facets to examine in developing the use case: (P1) How does the SCAN agent know when to initiate collaboration? (P2) What is the message structure necessary to handle collaboration? The message structure will be driven by the type of collaboration, therefore (P1) must be considered before (P2). For (P1), the following sequence of steps should take place in the Use Case: (S1) NASIC's communication to the satellite goes down (S2) NASICs SCAN agent attempts to diagnose the reasons (S3) Two causes are identified, a solar flare or a system problem (S4) Collaboration IDs are generated for each of the two causes (S5) NASICs SCAN agent sends a query to the other agents to collaborate (S6) Agents on behalf of the other agencies provide an affirmative response to the collaboration request (S7) The agencies are connected.

This protocol is motivated by Signaling System 7 (SS7) of the telephone network [13] which separates data from service. This property is particularly desirable, as it allows an agency to limit the amount of data it discloses to non-collaborating agencies. In the future, the topology of the SCAN agents will be further refined.

3.2 Basic model

From the perspective of a SCAN agent receiving collaboration requests, this agent is called the Collaboration Facilitator (CF). The requesting agent is called the CR. As stated previously, each SCAN agent will have a model of the other agents and agencies. The model couldl be used to derive such information as an arrival rate of resource requests. Based on the model of the requests coming in, the model couldl also predict a service rate of these requests. The model will also predict the benefits of each collaboration from each organization. Likewise, from the perspective of the CF, it is possible that a collaboration initiated by another agency will require collaboration with a third agency, and that only the CF and not the CR knows about this collaboration. In this case, it will NOT be the role of the CF to request collaboration with the third agency. This could reproduce the well-known "dining philosophers" problem from Operating Systems literature which is essentially that a bottleneck results when multiple processes compete for a limited set or resources. As a consequence of that bottleneck, system performance is degraded. Instead, the CF agent passes the information about the 3rd party back to the CR agent and the CR agent submits the request for collaboration to the third agency.

The motivation behind this model is that a SCAN agent receiving a collaboration request may be able to make decisions like: (1)"Well, technically I could collaborate you; after all I have enough resources to do it. However I know I'm about to get some high priority requests from important agencies soon, so... collaboration request denied." (2)"I can't collaborate with you right now. But my model says I'll be free in a few hours, do you want to book my time in advance?"

4 Multiagent models of interest

To model the type of problems described above, we need to consider the following requirements: sequential decisions, uncertainty regarding outcomes and other agents' decisions and decentralized control. Sequential decisions are needed because agents' decisions at the current time will affect the future outcomes. That is, suggesting to collaborate with one agency rather than another may have consequences in terms of time and resources required, affecting future collaboration possibilities. Uncertainty is present in these scenarios as well considering that information may be unavailable or outside forces could cause the situation to change. Also, users may decline collaboration and the actions of the other agents are often unseen, causing uncertainty about the other agents. The system must also be decentralized for similar reasons. Agents must make choices based solely on local information due to the lack of updates from other agents. This local information may also be uncertain or incomplete information about the human user or other humans and agents in the system.

This type of modeling can be accomplished with decision-theoretic approaches. In general, these representations have probabilities which represent the uncertainty in action outcomes and seek to maximize an objective function in an attempt to optimize the sequence of decisions by the agents. As agents are built for ever more complex environments, methods that consider the uncertainty in the system have strong advantages. Developing effective frameworks for reasoning under uncertainty is a thriving research

area in artificial intelligence and we discuss some of these approaches below. These models are briefly discussed below and summarized in Table 1.

4.1 DEC-POMDPs

A decentralized partially observable Markov decision process (DEC-POMDP) [4] can be defined with the tuple: $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O, T\rangle$ with I, a finite set of agents, S, a finite set of states with designated initial state distribution b_0, A_i, a finite set of actions for each agent, i P, a set of state transition probabilities: $P(s'|s, \boldsymbol{a})$, the probability of transitioning from state s to s' when the set of actions $\boldsymbol{a}$ are taken by the agents, R, a reward function: $R(s, \boldsymbol{a})$, the immediate reward for being in state s and taking the set of actions, $\boldsymbol{a}$ Ω_i, a finite set of observations for each agent, i, O, a set of observation probabilities: $O(\boldsymbol{o}|s', \boldsymbol{a})$, the probability of seeing the set of observations $\boldsymbol{o}$ given the set of actions $\boldsymbol{a}$ was taken which results in state s', T, a horizon or number of steps after which the problem terminates.

A DEC-POMDP involves multiple agents that operate under uncertainty based on different streams of observations. At each step, every agent chooses an action based on their local observation histories, resulting in a global immediate reward and a local observation for each agent. Note that because the state is not directly observed, it may be beneficial for the agent to remember its observation history. A *local policy* for an agent is a mapping from local observation histories to actions while a *joint policy* is a set of local policies, one for each agent in the problem. The goal is to maximize the total cumulative reward until the horizon is reached, beginning at some initial distribution over states. In the infinite-horizon problem, T is infinity and the decision making process unfolds over an infinite sequence of steps. In order to maintain a finite sum over the infinite-horizon, in these cases a discount factor, $0 \leq \gamma < 1$, is employed.

The DEC-POMDP model is very general, but has a very high complexity (NEXP-complete[1]). Algorithms for solving DEC-POMDPs also typically assume that all model parameters are known in advance and the solution is found in a centralized manner, providing policies that each agent can implement in a decentralized way. This model also assumes no communication (except as part of the observations) and full cooperation between the agents. The high complexity and large number of assumptions make the DEC-POMDP model currently inappropriate for the scenarios we are interested in for SCAN with a large number of agents with limited knowledge of the full problem model.

Communication and learning have also been studied in the DEC-POMDP model. Recent work has examined using communication for online planning in DEC-POMDPs [18]. This type of work could be useful in our context because communication is used when inconsistencies are detected between the agents. Unfortunately, it still requires knowledge of the full model and a DEC-POMDP to be solved for some number of steps. Rather than assuming the model is known, it can also be learned or policies can be learned directly [7, 16]. Direct policy learning may be an appropriate approach in that it does not require a model, but in order to calculate the gradient the model must be sampled. This sampling requires many instances and still is not guaranteed to converge to an equilibrium.

[1] This results in doubly exponential complexity as long as P!=NP.

4.2 DEC-MDPs and MMDPs

We can restrict DEC-POMDPs in various ways to simplify solution calculations. For instance, we can assume that each agent can fully observe its own local state, but not the global state. This is a form of the DEC-MDP model [2], which has been shown to have more efficient solutions in some cases, but in general has the same complexity as the general model (NEXP-complete) [4]. If we consider problems in which the agents have independent transitions and observations and a structured reward model (IT-IO DEC-MDPs), the complexity drops to NP-complete [3]. Recent work has shown that subclasses of DEC-POMDPs which have independent rewards, but dependent observations and transitions as well as those with certain structured actions retain the same complexity as the general problem [1]. Some other modeling assumptions and the resulting complexity are studied in [9], but none of these seems appropriate for our case as they continue to assume centralized solution methods and knowledge of the model.

DEC-MDPs can be further simplified by assuming that the state of the problem is fully observed by each agent, resulting in a multiagent MDP (MMDP) [5]. There has been work on efficient planning solutions for these problems (modeled as factored MDPs) [10], which allow algorithms to exploit independence between agents. While this is a useful quality (which is discussed in more detail in the context of ND-POMDPs), centralized solution methods are used and full model knowledge is required, limiting the applicability of these models to our scenario.

Model	Pros	Cons
DEC-POMDPs	Very rich model of cooperative agents	Assumes known model parameters, fully cooperative, high complexity
IO-IT DEC-MDPs	Somewhat rich, less complex than full model	Limiting model assumptions (interact only through rewards), Assumes known model parameters fully cooperative
MMDPs	Low complexity	Limiting model assumptions (centralized knowledge), Assumes known model parameters, fully cooperative
ND-POMDPs	Exploits locality, distributed solution, less complex than full model	Limiting model assumptions (very limited interaction), Assumes some model parameters fully cooperative
Graphical games	Allows self interest, exploits locality	Assumes some model parameters, not sequential
I-POMDPs	Allows self interest, exploits locality	Assumes known model parameters, high complexity

Table 1. Model pros and cons for distributed personal assistant domains.

4.3 ND-POMDPs

Another way of simplifying DEC-POMDPs is to assume agents interact in a limited way based on locality. This assumption often considers agents that can only interact with their neighbors and thus is more scalable in the number of agents in the problem. This has been studied using the networked distributed POMDP (ND-POMDP) model for the finite-horizon case [14] as well as a more general factored models [15, 17].

The ND-POMDP model is similar to the DEC-POMDP model except for the following: states can be factored $S = \times S_i \times S_u$ for each set of states for each agent i and an unaffected state set, actions and observations can be factored similarly with $A = \times A_i$ and $\Omega = \times \Omega_i$ for each agent, transition independence where the unaffected state and each agent transition independently $P(s'|s,a) = P(s'_u|s_u) \prod_i P(s'_i|s_i,a_i)$ and observation independence where $O(o|s',a) = \prod_i O_i(o'_i|s_i,a_i)$. Also, rewards can be broken up based on neighboring agents and summed as $R(s,a) = \sum_l (s_{l1},\ldots,s_{lk},s_u,\langle a_{l1},\ldots,a_{lk}\rangle)$ where l represents a group of $k = |l|$ neighboring agents. The ND-POMDP model also assumes that a the belief state (the state distribution estimate) is known by all agents.

A solution to an ND-POMDP can be computed in a distributed manner using message passing to converge to a local optimum. This approach can be very efficient in cases where the agents are only loosely connected. Nevertheless, strong assumptions are still made such as knowing the model in the local neighborhood and additive rewards across neighborhoods.

4.4 Game theoretic representations

If we assume that agents may have different interests that may not necessarily align perfectly (which is likely to be the case in many real-world scenarios), we could use a game theoretic approach. A model that retains the idea of agents loosely connect into neighborhoods is the graphical game [12]. This approach combats the tabular (and thus exponential) representation of n-agent games by using an undirected graph where two agents are connected if they can affect each others' payoffs. Assumptions in this model are that each agent can calculate a best response, which means that it knows the payoffs for the different actions given the other agents' choices. This is also not a sequential model, so all possible horizon T policies would have to be considered as actions, resulting in an exponential increase in action size and intractability to solve.

Sequential game theoretic models have also been studied. A generalization of DEC-POMDPs to the case where agents may have different rewards results in a partially observable stochastic game (POSG). Thus, if we assume POSGs with common payoffs for all agents, it is equivalent to the DEC-POMDP model [11]. This model relaxes the assumption that all agents are fully cooperative, but retains the intractability and knowledge of the model assumed by the DEC-POMDP. Another game theoretic view of multiagent POMDPs from the Bayesian perspective is the interactive POMDP (I-POMDP) model [8]. An extension of the I-POMDP model to graphical representations, allowing agent independence to be exploited is the interactive dynamic influence diagram (I-DIDs) [6]. While both of these models are interesting in that they can represent problems with estimates of other agent behavior based on observations that are seen, a full model is again assumed and complexity is at least as high as the corresponding DEC-POMDP models.

5 Dealing with private information

One weakness of the decision-theoretic models discussed in the previous section is the assumption that agents will have full knowledge of each others models. This is somewhat mitigated when only local agents are considered (as in an ND-POMDP), but even these local agents must share their information before a decentralized solution can be found. In a real-world scenario, such as SCAN, we cannot assume the agents will have knowledge of the other agent's model: states, actions, observations, let alone their transition and observation probabilities or rewards values.

There may be privacy or security reasons for not sharing this information. For instance, individuals may not want to disclose their full schedules or preferences and organizations do want to share proprietary information. More specifically, Chris may have a preference for meeting with Paul over Nathan which he would rather not divulge (especially to Nathan!). Also, there may be several other projects that a person is working on that have no direct relationship to a possible collaboration between entities, but they be affected in some way. For example, I may not want to share my schedule for all projects that I am working on, but I would consider changing my schedule to collaborate on a particularly important or interesting project. This could change depending on who is working on the project, when it is being scheduled, different travel locations etc. Nevertheless, planning must be accomplished based on this limited model information. To represent these problems, we describe a new model, the *shared MDP*, which is an initial step towards dealing with shared and private information.

5.1 Shared MDP model

We present the shared MDP model, but this could be extended to the POMDP case. In a shared MDP, each agent, i, consists of an MDP with S_i, A_i, P_i, R_i and T_i, the states, actions, transitions, rewards and horizon for agent i's model. There may be some set of states which are *shared* by a set of agents. For agent i, we denote the set of shared states as S_i^s. In this case, the transitions and rewards for the shared states depend on all agents that are in the shared states at that time.

More formally, we consider $S = \cup_i S_i$ the full set of states for all agents, $D(s) = \{i | s \in S_i\}$ the possible set of decision-makers at state s as well as joint reward $R(s, \boldsymbol{a_{D(s)}})$ and transitions for each agent i $P_i(s'|s, \boldsymbol{a_{D(s)}})$. We define a *private state* as an agent's state that is not shared by any other agent. That is, state s is private for agent i if $D(s) = i$. Note that $D(s)$ represents all agents that have state s in their MDP, but some or all of these agents may be in a private state at the given time and therefore do not affect the rewards and transitions of the other agents. In these cases, we can replace the action of the agents in private states with a dummy action or omit them altogether.

Proposition 1. *A shared MDP with no shared states is equivalent to a set of independent MDPs.*

Proof is straightforward and thus omitted.

Proposition 2. *A shared MDP with no private states and common transition functions is equivalent to an MMDP.*

Proof is straightforward and again omitted. In general, agents will have full knowledge of the (common) state of the system as well as the joint transitions and rewards. This is also the case if agents' MDPs are subsets of other agents' MDPs.

The interesting (and realistic) case occurs when each agent has a set of private states that are not shared with the others. A shared MDP could also be solved centrally as one large MDP, but this requires sharing private and secure information with a central agency (by communicating the details of the private model as well as the shared one). A central approach would also introduce a single point of failure and make it more difficult to add or remove agents over time. As a result, we explore fully decentralized solution methods.

5.2 Example shared MDP

An example shared MDP with two agents is shown in Figure 2. Here, $S_1 = \{s^1, s^2\}$ and $S_2 = \{s^2, s^3\}$, resulting in shared state s^2 and private states s^1 and s^3 for agents 1 and 2 respectively. We also assume $A_1 = A_2 = \{a^1, a^2\}$.

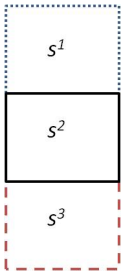


Fig. 2. Shared MDP with two agents. Agent 1 contains states 1 and 2, while agent 2 contains states 2 and 3. State 2 is a shared state.

The dynamics for agent 1 in the private state will be that a^1 causes the agent to stay in the private state, $P(s^1|s^1, a^1) = 1$, while a^2 causes the agent to transition to the shared state $P(s^2|s^1, a^2) = 1$. Agent 2 has the same dynamics $P(s^3|s^3, a^1) = 1$ and $P(s^2|s^3, a^2) = 1$. Note the agent subscript is omitted for private states. In the shared state, if both agents take the same action, each agent transitions back to its private state. That is, $P_1(s^1|s^2, a_1^1, a_2^1) = 1$, $P_1(s^2|s^2, a_1^1, a_2^2) = 1$, $P_1(s^2|s^2, a_1^2, a_2^1) = 1$, $P_1(s^1|s^2, a_1^2, a_2^2) = 1$. The same can be written for agent 2's P_2. When only one agent is in the shared state, its dynamics are the same as those for the private state, $P_1(s^1|s^2, a^1) = 1$, $P_1(s^2|s^2, a^2) = 1$, $P_2(s^3|s^2, a^1) = 1$ and $P_2(s^2|s^2, a^2) = 1$.

If $R_1(s^1, a^1) = R_1(s^1, a^2) = 10$, $R_2(s^3, a^1) = R_2(s^3, a^2) = -10$ and for both agents, $R(s^2, a_1^1, a_2^1) = R(s^2, a_1^1, a_2^2) = R(s^2, a_1^2, a_2^1) = R(s^2, a_1^2, a_2^2) = 0$, and when only one agent is in s^2 $R(s^2, a^1) = R(s^2, a^2) = -2$ then there is no cooperative solution to this problem in which both agents attain their highest values. This is because agent 1 will gain higher reward for being in its private state and thus will attempt to

coordinate with agent 2 to choose the same action to transition accordingly. On the contrary, agent 2 will gain higher value for staying in the shared state and thus will attempt to choose a different action than agent 1. These differences can be seen in the optimal policies for each agent's MDP. Assuming an infinite horizon problem with discount of 0.9, agent 1 produces a value of 100 starting in s^1 by always choosing action a^1, while if both agents start in s_2 a value of 90 can be achieved if both agents choose the same action on the first step and then agent 1 chooses a^1 thereafter. For each step that the same action is not chosen, value is lost by agent 1. This is in contrast to agent 2, who achieves at most -9 after starting in s^2 and cooperating with agent 1 to choose the same action, while it could attain a value of 0 if both agents always chose different actions. Starting in s^3, agent 2 can attain a value of -10 by choosing a^2 and choosing the opposite action as agent 1 if that agent is in the shared state s^2. If agent 1 is not in the shared state for any other step, a value of -28 can be attained by always choosing a^2.

5.3 Solutions for shared MDPs

It can be shown that in the shared MDP example above, starting from s^2, agent 1 can ensure that it receives at least 80 by randomly choosing either action in s^2. A similar worst case value can be found for agent 2, which again would randomly choose actions in s^2 when agent 1 is present to stay in that state as long as possible. This solution can be found by transforming a shared MDP into a competitive problem. This is appropriate since the presence of private states may cause each agent's value to be different for taking the same set of actions in the same state (since the value of an action in a state depends on on all subsequent rewards that can be obtained, shared or private).

To produce a solution to any shared MDP, it is necessary to combine solutions for the private and shared aspects of each agent's MDP. Algorithm 1 shows how a policy for agent i can be found. This algorithm constructs and solves a game (for the Nash equilibria, NE) from solving MDPs that consider all possible policies for all agents for shared states of the shared MDP. That is, it first constructs a PolicySet, which consists of all mappings from shared states S_i^s of agent i to actions for all agents which also contain that shared state. For each agent and each of these policy mappings, an MDP is solved which consists of the private states of the agent's MDP and the fixed policy given by PolicySet for the shared states. These MDP values are then used to determine a Nash equilibrium (if we assume a unique one is found) which serves as the policy for agent i.

In the case that multiple equilibria are found, coordination mechanisms or equilibria refinements can be used. In order to coordinate, the agents could share the equilibria found for the shared states to determine a single one that can be played (ensuring that policies for the shared states are known to all agents). Also, more efficient ways for finding policies in Algorithm 1 can be utilized. This could include not considering all possible policies for the other agents in PolicySet, but rather some subset, which could be based on heuristic value. If matching equilibria cannot be found, the algorithm could be repeated (with different parameters) until a solution is found.

In practice, we are interested in finding a pure strategy Nash equilibrium as a deterministic solution is preferred by our users. This will often allow us to determine a

Algorithm 1 SOLVESHAREDMDP(i)

```
PolicySet ← {}
for all s, ∈ S_i^s do
    Policy(s) ← null
    for all j, ∈ D(s) do
        for all a, ∈ A_j do
            PolicySet.Add(s_j,a)
for all p, ∈ PolicySet do
    Value(S_i^s,p) ← SolvePrivateMDP(S_i^s, p)
Policy ← SolveForNE(Value)
return Policy
```

solution to the problem much more quickly and only resort to using Algorithm 1 when a pure strategy NE cannot be found. We will attempt to discover a NE with iterated best response as shown in Algorithm 2. That is, each agent begins with a random policy (or one previously found) for the shared states and, while keeping the other agents' policies fixed, an agent determines a best response for its MDP. This new best response policy for the shared states is now fixed, along with other agents' policies, while the next agent determines a best response. This continues until no agent changes its policy, resulting in a Nash equilibrium. Unfortunately, it might be the case that there are no pure strategy NE in the given problem, causing the agent policies to continue changing forever. Currently, the algorithm checks for these oscillations by making sure the total number of changes by the agents is less than the total number of possible policies (m^n where m is the number of strategies and n is the number of agents). More sophisticated analysis of oscillation could also be used to determine when Algorithm 1 should be used to find a mixed strategy. In practice, it will often be the case that several pure strategy NE exist, allowing a solution to be found quickly.

Algorithm 2 SOLVESHAREDMDPDETERMINISTIC

```
Policies ← RandomPols()
Converged ← 0
Oscillate ← false
while Converged < n − 1 or Oscillate < possSolutions do
    for all i ∈ I do
        Policies(i) ← BestResponse(Policies(−i))
        if IsChanged(Policies(i)) then
            Converged ← 0
        else
            Converged++
        Oscillate++
if Converged=n-1 then
    return Policies
else
    return false
```

6 Conclusion

In this paper, we discussed a real-world domain for facilitating collaboration between organizations and people, the SCAN proxy agents. We described the characteristics of this domain which require sequential and decentralized decision-making. Various decision-theoretic models for representing these problems are presented along with their shortcomings in this domain. In order to begin to address these shortcomings, we present one approach for representing and solving problems with private and shared information as a shared MDP. This is an initial step towards providing decentralized solutions to sequential collaboration problems with private information. In the future, we are interested in further extending decision-theoretic models so they can be applied in this context. This work will include other model assumptions that better fit the real-world data that is generated during this project. These algorithms will be implemented and tested in the SCAN domain.

References

1. Allen, M., Zilberstein, S.: Complexity of decentralized control: Special cases. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I., Culotta, A. (eds.) Advances in Neural Information Processing Systems, pp. 19–27. 22 (2009)
2. Becker, R., Lesser, V., Zilberstein, S.: Decentralized Markov Decision Processes with Event-Driven Interactions. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 302–309. New York, NY (2004)
3. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Solving transition-independent decentralized Markov decision processes. Journal of AI Research 22, 423–455 (2004)
4. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of Markov decision processes. Mathematics of Operations Research 27(4), 819–840 (2002)
5. Boutilier, C.: Sequential optimality and coordination in multiagent systems. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 478–485. Stockholm, Sweden (1999)
6. Doshi, P., Zeng, Y., Chen, Q.: Graphical models for interactive POMDPs: Representations and solutions. Journal of Autonomous Agents and Multi-Agent Systems 18 (2009)
7. Dutech, A., Buffet, O., Charpillet, F.: Multi-agent systems by incremental gradient reinforcement learning. In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence. pp. 833–838 (2001)
8. Gmytrasiewicz, P.J., Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research 24, 24–49 (2005)
9. Goldman, C.V., Zilberstein, S.: Decentralized control of cooperative systems: Categorization and complexity analysis. Journal of Artificial Intelligence Research 22, 143–174 (2004)
10. Guestrin, C., Koller, D., Parr, R.: Multiagent planning with factored MDPs. In: Advances in Neural Information Processing Systems, pp. 1523–1530. 15 (2001)
11. Hansen, E.A., Bernstein, D.S., Zilberstein, S.: Dynamic programming for partially observable stochastic games. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence. pp. 709–715. San Jose, CA (2004)
12. Kearns, M., Littman, M.L., Singh, S.: Graphical models for game theory. In: Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (2001)

13. Modarressi, A., Skoog, R.: Signalling system no. 7: A tutorial. IEEE Communications Magazine July (1990)
14. Nair, R., Varakantham, P., Tambe, M., Yokoo, M.: Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (2005)
15. Oliehoek, F.A., Spaan, M.T.J., Whiteson, S., Vlassis, N.: Exploiting locality of interaction in factored Dec-POMDPs. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems. Estoril, Portugal (2008)
16. Peshkin, L., Kim, K.E., Meuleau, N., Kaelbling, L.P.: Learning to cooperate via policy search. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence. pp. 489–496 (2000)
17. Witwicki, S.J., Durfee, E.H.: Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In: Proceedings of the Twentieth International Conference on Automated Planning and Scheduling. Toronto, Canada (2010)
18. Wu, F., Zilberstein, S., Chen, X.: Multi-agent online planning with communication. In: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling. Thessaloniki, Greece (2009)