A Near-Optimal Node-to-Agent Mapping Heuristic for GDL-Based DCOP Algorithms in Multi-Agent Systems

Md. Mosaddek Khan
School of Electronics and Computer Science
University of Southampton, Southampton, UK
mmk1g14@ecs.soton.ac.uk

Long Tran-Thanh
School of Electronics and Computer Science
University of Southampton, Southampton, UK
ltt08r@ecs.soton.ac.uk

William Yeoh
Department of Computer Science and Engineering
Washington University in St. Louis, St. Louis, USA
wyeoh@wustl.edu

Nicholas R. Jennings
Departments of Computing and Electrical and Electronic Engineering, Imperial College London, London, UK
n.jennings@imperial.ac.uk

ABSTRACT

Distributed Constraint Optimization Problems (DCOPs) can be used to model a number of multi-agent coordination problems. The conventional DCOP model assumes that the subproblem that each agent is responsible for (i.e. the mapping of nodes in the constraint graph to agents) is part of the model description. While this assumption is often reasonable, there are many applications where there is some flexibility in making this assignment. In this paper, we focus on this gap and make the following contributions: (1) We formulate this problem as an optimization problem, where the goal is to find an assignment that minimizes the completion time of the DCOP algorithm (e.g. Action-GDL or Max-Sum) that operates on this mapping. (2) We propose a novel heuristic, called MNA, that can be executed in a centralized or decentralized manner. (3) Our empirical evaluation illustrates a substantial reduction in completion time, ranging from 16% to 40%, without affecting the solution quality of the algorithms, compared to the current state of the art. In addition, we observe empirically that the completion time obtained from our approach is near-optimal; it never exceeds more than 10% of what can be achieved from the optimal node-to-agent mapping.

KEYWORDS

Distributed Problem Solving; DCOP; GDL; Node-to-Agent Mapping

ACM Reference Format:
Md. Mosaddek Khan, Long Tran-Thanh, William Yeoh, and Nicholas R. Jennings. 2018. A Near-Optimal Node-to-Agent Mapping Heuristic for GDL-Based DCOP Algorithms in Multi-Agent Systems. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to coordinate the assignments of values to their variables in such a way that maximizes their aggregated utility [7, 16, 18]. This model can be applied to solve a number of multi-agent coordination problems including distributed meeting scheduling [15], sensor networks [4, 28], multi-robot coordination [26], coalition formation [3] and smart homes [9, 20]. The problems are formulated as constraint networks that are often represented graphically using one of the following representations: junction trees [1, 21], factor graphs [5, 12] or Depth First Search (DFS) trees [16, 18]. In all of these representations, nodes (i.e. variables and/or factors depending on the graphical representation) are being held[1] by the agents participating in the optimization process.

The conventional DCOP model assumes that the mapping of nodes to agents is part of the model description. In other words, the nodes that each agent holds are given as an input. This assumption is reasonable in many applications where there are obvious and intuitive mappings – for example, in a smart home scheduling problem [9], agents correspond to the different smart homes, and variables (i.e. nodes) correspond to the different smart devices within each home. In this case, the agent controls all the variables that map to the devices in its home. However, in other applications, there may be more flexibility in the mapping of nodes to agents. For example, imagine an application where a team of unmanned aerial vehicles (UAVs) need to coordinate with each other to effectively survey an area. In this application, agents correspond to UAVs and variables correspond to the different zones in the area to be surveyed. The domain for each variable may correspond to the different types of sensors to be used and/or the different times to survey the zone. Since a UAV can survey any zone, there are multiple possible assignments of zones to UAVs. That is, there are multiple possible mappings of variables (i.e. nodes) to agents.

While it is possible to arbitrarily choose a mapping and run any off-the-shelf DCOP algorithm to solve the problem, choosing a good mapping is important as it can have a significant impact on an algorithm's completion time (as we shall discuss in the following section). However, choosing an optimal mapping may be prohibitively time consuming as this is an NP-hard problem problem, as shown by Rust, Picard and Ramparany [20]. In that paper, they introduced a simple heuristic of node-to-agent mapping that is specifically tailored to their smart-home application, called Smart Environment Configuration Problem (SECP). Therefore, this method is not applicable to other DCOP settings. Considering these issues, coupled with the fact that this step is only a preprocessing step prior to the actual DCOP algorithm, we develop a generic heuristic algorithm to address the problem of node-to-agent mapping in DCOPs.

[1] The agents act (i.e. generate and transmit messages) on behalf of the nodes they hold.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

In more detail, this paper advances the state of the art as follows. (1) We propose a new time-efficient heuristic to determine a near-optimal **M**apping of **N**odes to the participating **A**gents (**MNA**). **MNA** is a preprocessing step that works prior to executing the optimization process of a GDL-based DCOP algorithm. Specifically, **MNA** can be executed in a centralized or decentralized manner, depending on the application at hand. As a preprocessing step, **MNA** does not alter any internal process of the original DCOP algorithm; therefore, it does not have any impact on its solution quality. Additionally, the decentralized version of **MNA** specifically caters for scenarios where the graphical representation experiences change(s) during runtime. (2) We empirically evaluate the performance of **MNA** in terms of completion time, and show that it performs at a level of around $90\% - 100\%$ of the optimal mapping, which is computationally infeasible to obtain in practice. (3) Our results also show a speed-up of $16\% - 40\%$ compared to the state of the art, meaning a message passing algorithm can perform $1.2 - 1.7$ times faster when using node-to-agent mappings generated by **MNA**.

The remainder of this paper is structured as follows. In Section 2, we formulate this particular phase of node-to-agent mapping as an optimization problem, where the objective is to obtain an assignment that reduces the completion time of a GDL-based DCOP algorithm that operates on this mapping. In Section 3, we discuss the details of both the centralized and decentralized versions of **MNA**. Afterwards, Section 4 reports the empirical evaluation of our approach as opposed to the current state-of-the-art, and Section 5 concludes.

2 BACKGROUND AND PROBLEM FORMULATION

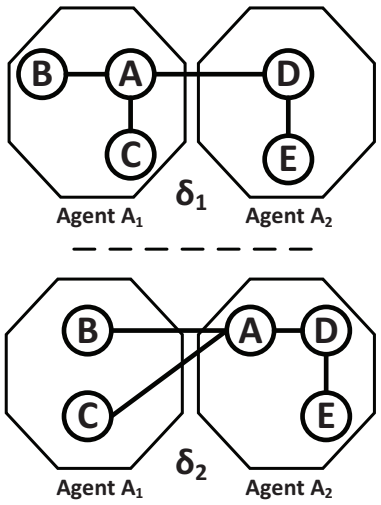
A DCOP is defined by a tuple $\langle X, D, F, A, \delta \rangle$ [16], where X is a set of discrete variables $\{x_0, x_1, \ldots, x_m\}$ and $D = \{D_0, D_1, \ldots, D_m\}$ is a set of discrete and finite variable domains. Each variable x_i can take its value from the domain D_i. F is a set of constraints $\{F_1, F_2, \ldots, F_n\}$, where each $F_i \in F$ is a function dependent on a subset of variables $\mathbf{x}_i \in X$ defining the relationship among the variables in $\mathbf{x}_i$. Thus, the function $F_i(\mathbf{x}_i)$ denotes the value for each possible assignment of the variables in $\mathbf{x}_i$. The dependencies between the variables and the functions are often graphically represented by a constraint graph such as a junction tree or a factor graph, where the nodes (i.e. variables and/or functions) of the corresponding graphical representation G are being held by a set of agents $A = \{A_1, A_2, \ldots A_k\}$. This mapping of nodes to agents is represented by $\delta : \eta \rightarrow A$. Here, η stands for the set of nodes within the constraint graph G. As a result of the mapping represented by δ, we get a partition $P(A)$ of $k = |A|$ sub-graphs ($i.e.$ $G_1, G_2, \ldots, G_k$) from G, where each $G_j \in G$ is held by the agent $A_j \in A$ (Equation 1).

$$P(A) \leftarrow \bigcup_{j=1}^{k} G_j \mid \forall j' \neq j : (G_j \cap G_{j'}) = \emptyset \tag{1}$$

Within this model, a DCOP algorithm (e.g. Action-GDL [24], Max-Sum [5] or Bounded Max-Sum [19]) operates directly on G by passing messages among the nodes $\eta \in G$ to have each agent assign values to its associated variables from their corresponding domains. The aim is to maximize (or minimize) the aggregated global objective function which eventually produces the value of each variable, $X^* = \arg\max_X \sum_{i=1}^{n} F_i(\mathbf{x}_i)$. As already mentioned, in this work, we specifically concentrate on GDL-based DCOP algorithms such as Action-GDL, Max-Sum and Bounded Max-Sum. In such algorithms, to compute a message for a particular neighbour, a node takes into account the messages from its neighbours along with its own utility. Thus, a number of nodes initially start generating (i.e. computation) and then sending (i.e. communication) messages, each of which we jointly denote as a single event. That means, an event involves both the computation and communication of a certain message. In this process, the completion of certain events might trigger one or more new events to be initiated. Thus, the total message passing procedure will complete when each node receives messages from all of its neighbours, such that all the running events are completed without initiating any new events. The dependencies among the events during the message passing process can be seen as an event-based dependency graph $E_G(\mathbb{A}, P)$, where $\mathbb{A}$ is the specific DCOP algorithm deployed and P is the partition obtained from Equation 1. Formally, let E be the set of events $\{E_1, E_2, \ldots, E_l\}$ of $E_G(\mathbb{A}, P)$. Here, the weight of an edge $E_i \rightarrow E_j$ between two events E_i and E_j represents the time required to complete event E_i. Finally, the longest path cost of all existing event pairs is the total completion time $T(\mathbb{A}, P)$ for a given graphical representation of a DCOP (Equation 2). Here, the function $v(E_i, E_j)$ represents the time elapsed (i.e. path cost) between the starting of the event E_i and the end of the event E_j.

$$T(\mathbb{A}, P) = \max_{\forall_{E_i, E_j} \in E_G(\mathbb{A}, P)} v(E_i, E_j) \tag{2}$$

In this formulation, without loss of generality, we assume each agent possesses its own memory and a separate processing unit.[2] Here, on behalf of the sending node of an event, the holding agent generates and then sends the message to the receiving node. The



Computation Cost	
Node	Time-unit
A	25
B	0
C	0
D	12
E	0

Communication Cost	
Type	Time-unit
Inter-agent	7
Intra-agent	2

Figure 1: Two sample mappings of nodes $\{A, B, C, D, E\}$ of a constraint graph to agents A_1 and A_2. In the figure, nodes are denoted by circles and agents as octagons.

[2] In a multi-processing capable setting, each processing unit with separate memory can be considered as an agent.

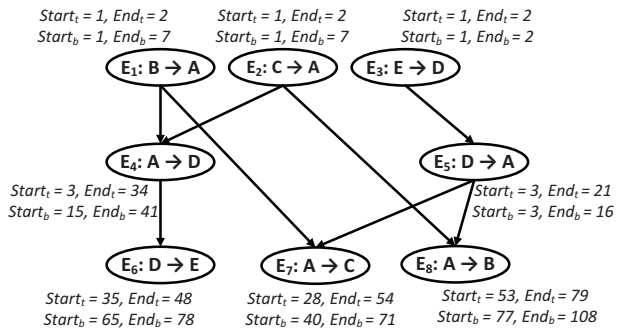


Figure 2: Event-based dependency graph for the constraint graph of Figure 1.

sending node and its corresponding receiving node can either be held by the same agent or by two different agents. The time required to send a message in the former case can be termed the intra-agent communication cost and the latter the inter-agent communication cost. The former is typically less expensive in terms of communication cost than the latter [8, 22]. This is because it requires less time for an agent to take a message from its local memory than from a memory belonging to a different agent. Moreover, since an agent has a single processing unit, it cannot compute more than one message at a time. However, it can compute a message while transmitting another one and vice versa. As a consequence, allowing an agent to hold too many nodes eventually increases the waiting time for the nodes within the agent. Considering this trade-off, the ultimate objective is to minimize the completion time $T(\mathbb{A}, P)$ of a message passing algorithm $\mathbb{A}$ by providing an efficient mapping of nodes to agents (Equation 3).

$$P^* = \arg\min_{P} T(\mathbb{A}, P) \tag{3}$$

Figure 1 illustrates two sample assignments of a constraint graph having five nodes $\{A, B, C, D, E\}$ between two agents A_1 and A_2. On the one hand, two sets of nodes $\{A, B, C\}$ and $\{D, E\}$ are being held by the agents A_1 and A_2 respectively in the mapping δ_1, depicted at the top of Figure 1. On the other hand, A_1 holds nodes $\{B, C\}$ and A_2 holds nodes $\{A, D, E\}$ in the mapping δ_2, shown at the bottom of that figure. Additionally, the message computation cost of each node and the message transmission/communication cost for the edges in terms of time-units are given in the tables on the right side of Figure 1. As can be seen, the computation cost of node A is 25 time-units, meaning node A requires 25 time-units to generate a message for any one of its neighbours. In this example, the inter-agent and the intra-agent communication cost is 7 and 2 time-units, respectively. Thus, the sending node A requires 7 time-units to send a message to the receiving node D when both A and D are being held by different agents (δ_1). Otherwise, the same message takes 2 time-units (δ_2).

The reason why the efficient mapping of node-to-agent is significant can be clearly seen from Figure 2, where we generate an event-based dependency graph of the message passing for the exemplar constraint graph shown in Figure 1. Here, the starting and finishing time of each event are represented by $Start_t/Start_b$ and End_t/End_b respectively, where t stands for the mapping δ_1 and b corresponds to δ_2. Finally, the largest value of End_t and End_b represents the completion time of the constraint graph based on the mappings δ_1 and δ_2, respectively. In this particular example, we get 8 events: $\{E_1, E_2, \ldots, E_8\}$. For instance, event E_4 stands for the summation of the computation and the communication time of the message sending from A to D and the event E_4 can only initiate after events E_1 and E_2 have finished, that is when node A receives messages from nodes B and C. It is worth mentioning that if the holding agent of node A (i.e. the sending node of event E_4) is already computing a message for another node, then E_4 has to wait until the agent finishes computing the message, even if E_1 and E_2 have finished. Significantly, the degrees of nodes A and D are higher than those of other nodes in the constraint graph. As such, they require substantially more time-units to generate each of their messages. In the mapping δ_2, both A and D are being held by agent A_2. This potentially leads to a situation where the nodes of A_2 have to wait for a long period of time, even if the events they depend on have finished. In this worked example, events E_7 and E_8 have to wait for an additional 24 and 61 time-units respectively, even though they are ready to compute (δ_2). On the other hand, the waiting times are 7 and 32 time-units respectively in δ_1 due to the fact that the higher degree nodes A and D are held by two different agents. As a result, we observe that the completion time of a DCOP algorithm for the mapping δ_1 is 79 time-units, and 108 time-units for δ_2. Thus, even for a small constraint graph of 5 nodes, it is possible to save around 27% of completion time through an efficient node-to-agent mapping.

However, finding an optimal mapping is an NP-hard problem [20]. Consider an example where a constraint graph of 25 nodes have to distribute among 8 agents. In this case, there are $1,081,575$ possible uniform mappings. In addition to that, we cannot ignore the possibility of getting better results from a non-uniform assignment. Even though the search space can be reduced by giving preference to the contiguous nodes being held by the same agent, the number is still significant (see empirical results). Furthermore, the optimal mapping is completely dependent on the structure of the graph, so it is not possible to predict such mapping in advance based on prior information. Under such circumstances, finding an optimal mapping is not practicable for large multi-agent settings. This leads us to the **MNA** heuristic detailed in the following section.

3 THE MNA HEURISTIC

Considering the observations made in the previous section, **MNA** specifically aims to find mappings where nodes with high degrees are held by different agents. In other words, the objective is to obtain a node-to-agent mapping for a DCOP, where nodes with higher computational requirements for producing their messages do not end up being held by the same agent. At the same time, it is important to ensure that the mapping process itself is not prohibitively expensive in terms of time consumption. To this end, we propose two versions of **MNA**, centralized and decentralized, each of which is discussed in Sections 3.1 and 3.2, respectively.

3.1 Centralized Version of MNA

The complete process of **MNA**'s centralized version is detailed in Algorithm 1. As aforementioned, it aims to reach a point where no

Algorithm 1: **MNA** (G, η, A, $\mathbb{A}$)

Input: G is the corresponding graphical representation of a DCOP consisting of a set $\eta = \{\eta_1, \eta_2, \ldots \eta_N\}$ of N nodes and $A = \{A_1, A_1, \ldots A_k\}$ is the set of k agents participating in the optimization process, where $k <= N$. $\mathbb{A}$ stands for the deployed GDL-based DCOP algorithm.

Output: Mapping δ of the nodes of η to their associated agents A (i.e. $\delta : \eta \rightarrow A$), so that overall completion time can be minimized. Note that, each node can be held by a single agent; however, each agent can hold several nodes.

1 Let $deg = \{deg(\eta_1), deg(\eta_2), \ldots, deg(\eta_N)\}$ be the set where each $deg(\eta_i) \in deg$ stands for the degree/number of connected nodes of η_i

2 D is a set of domains $\{D_1, D_2, \ldots, D_N\}$, where each $D_i \in D$ is a finite set containing the values from which its associated node η_i has to get its preferred value

3 $uniformVal \leftarrow N/k$

4 **if** $(deg(\eta_i) - deg(\eta_{i'}) == 0) \wedge (|D_i| - |D_{i'}| == 0)$, $where\ \forall \eta_i, \eta_{i'} \in \eta, \forall D_i, D_{i'} \in D$ **then** // Contiguous uniform node-to-agent mapping, when the nodes possess similar degree and equal domain size.

5 **return** $\delta_{uniformVal} : \eta \rightarrow A$

6 **else**

7 $\lambda \leftarrow k\text{-}largestNodes(\text{G}, \eta, k)$ // Find a set λ of k largest nodes from η in terms of degree. Use the domain size of the connected nodes in case of a tie.

8 $\delta : \lambda \rightarrow A$ // Distribute the nodes of $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ to A such that each agent holds a single node.

9 $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ are the control points of the graph G

10 **foreach** *node* $\eta_i \in \eta \setminus \lambda$ **do** // Distribute non control point nodes.

11 $\lambda_{cp} \leftarrow minDistance(\text{G}, \eta_i, \lambda, uniformVal),\ where\ \lambda_{cp} \in \lambda$ // Call Algorithm 2: choose the suitable control point λ_{cp} for the node η_i.

12 $\delta : \eta_i \rightarrow \lambda_{cp}.A_{cp}$ // allocate η_i to the agent A_{cp} that holds the control point λ_{cp}.

13 **return** $\delta : \eta \rightarrow A$

two high-degree nodes are held by the same agent. Subsequently, a suitable agent is picked for each of the remaining nodes of a DCOP graphical representation based on this initial assignment. **MNA** operates directly on the corresponding graphical representation G of a DCOP that is going to be solved by deploying a GDL-based algorithm $\mathbb{A}$. Here, G consists of a set $\eta = \{\eta_1, \eta_2, \ldots \eta_N\}$ of N nodes and a set $A = \{A_1, A_1, \ldots A_k\}$ of k agents. At the end, Algorithm 1 returns $\delta : \eta \rightarrow A$, that is the mapping δ of the nodes η to their associated agents A. In line 1, a set $deg = \{deg(\eta_1), deg(\eta_2), \ldots, deg(\eta_N)\}$ represents the number of connected neighbours of the nodes in η. More specifically, the function $deg(\eta_i) \in deg$ stands for the number of neighbours of the node $\eta_i \in \eta$, and it also provides information regarding how many incoming messages are required to produce each of η_i's outgoing messages, taking the deployed algorithm $\mathbb{A}$ into consideration. Then, line 2 presents the set of domains D, and each $D_i \in D$ is a finite set containing the values from which its associated node η_i has to take its preferred value. It is clearly illustrated in the example of the previous section that the degree of each node and the domain sizes of the connected neighbouring nodes contribute significantly in determining the overall completion time for a particular mapping. To be exact, the computation cost of the node η_i in terms of time corresponds to the values of $deg(\eta_i)$ and D_i. In the worked example of Figure 1, the degrees of node A and B are 3 and 1, respectively. Therefore, node A has to consider the messages of at least two nodes along with its own utility to generate a message for any of its neighbours. Moreover, the time required to generate a message is highest for node A, as its degree is higher than that of any other nodes. On the other hand, node B only needs to send a message to its only neighbouring node A. Consequently, for B to be able to generate that message, it does not need to rely on receiving any other message. As a result, B can immediately generate the message based on its local utility or often this is a pre-defined initial message. Thus the computation cost of B is negligible. Afterwards, line 3 computes the value of $uniformVal$, which is the ratio of the number of nodes N and the number of agents k in G.

It is noteworthy that the problem of node-to-agent mapping becomes trivial if all the nodes possess similar degrees and equal domain size. In this case, we can uniformly distribute the nodes among the agents by giving preference to the contiguous nodes being held by the same agent (lines 4–5). Nevertheless, this is not the case for most DCOP applications, rather it is common to have nodes with dissimilar degrees and domain size [11, 13]. This phenomenon, particularly, accounts for the differences in completion time for various possible mappings of nodes to agents. Specifically, lines 6 – 13 of the algorithm concentrate on this issue. Now, the function $k\text{-}largestNodes(G, \eta, k)$ finds the k largest nodes from η in terms of degree. In case of a tie, it uses larger domain size, then records them to a set $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ (line 7). As a result, we get top k nodes with the highest degrees in G that require more time-units to compute each of their messages. At this point, line 8 allocates each node $\lambda_i \in \lambda$ to the different agents of A, and **MNA** defines each of these nodes as a control point (explained shortly) of the constraint graph G (line 9). In other words, the set $\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ of k high-degree nodes are going to act as the control points, each of which is exclusively held by one of the k agents of A. In the

Algorithm 2: minDistance($G, \eta_i, \lambda, uniformVal$)

Input: λ is a set of control points of the graph G, η_i is a non-control point node of G to be associated with one of the control point nodes of λ and $uniformVal$ is obtained from line 3 of Algorithm 1.

Output: $\lambda_m \in \lambda$, the corresponding control point for η_i.

1 $\lambda' \leftarrow \lambda$
2 $\lambda_m \leftarrow sPath(G, \eta_i, \lambda')$
3 **if** $p(\lambda_m, A_m) < uniformVal$ **then** // when the agent A_m corresponds to λ_m holds fewer nodes than the value of $uniformVal$.
4 | **return** λ_m
5 **else**
6 | $\lambda' \leftarrow \lambda' \setminus \lambda_m$
7 | **if** $\lambda' \neq \emptyset$ **then**
8 | | **go to** line 2
9 | **else**
10 | | $\lambda_m \leftarrow alt\_sPath(G, \eta_i, \lambda)$ // assign η_i to the closest control point that does not currently associates the most number of non-control point nodes among λ.
11 | | **return** λ_m

example of Figure 1, the agents A_1 and A_2 are participating in the optimization process, hence the value of k is two. Therefore, we need to find two control points from the set of nodes: $\{A, B, C, D, E\}$. In this particular instance, **MNA** picks A and D as the control points as they posses degrees that are higher than those of the other nodes, and they should be held by those two different agents. Let A and D be held by agents A_1 and A_2, respectively. This is significant because it assures that no two high-degree nodes will be held by the same agent, which is the biggest cause of an increase in the waiting time (as discussed in the previous section).

At this point, the for loop of lines $10 - 12$ associates the rest of the nodes that are not the control points (i.e. $\eta \setminus \lambda$), to their corresponding agents. In so doing, we utilize the concept of Fortune's algorithm to generate the Voronoi diagram [10]. Notably, a Voronoi diagram is a partitioning of a plane into regions based on the distance to a specific subset of points of the plane. This subset of points, denoted as control points, is specified beforehand. For each of the control points, Fortune's algorithm generates a corresponding region consisting of all points closer to the control point than to others. In other words, given a set of control points in a plane, Fortune's algorithm specifically finds the associated control points for the rest of the points on that plane, based on the nearest Euclidean distance at the worst case cost of only $O(N \log N)$ time. Here, the function $minDistance$(G, $\eta_i, \lambda, uniformVal$), detailed in the pseudo-code of Algorithm 2, takes as input a non-control point node η_i, the subset λ of η that acts as the control points and previously computed $uniformVal$, and then finds a suitable control point $\lambda_{cp} \in \lambda$ for η_i (line 11 of Algorithm 1).

The function is inspired by the method employed by Fortune's algorithm to obtain the appropriate control points for all such non-control point nodes. However, unlike Fortune's algorithm, which uses only the shortest Euclidean distance as the metric to choose the suitable control point for a node, **MNA** uses different criteria. This is because we have to deal with a graphical representation instead of a plane. In more detail, in line 2 of Algorithm 2, $sPath(G, \eta_i, \lambda')$ finds such a control point $\lambda_m \in \lambda'$ for η_i that possesses the shortest path from η_i within the constraint graph G, and G is considered as an unweighted graph during this process. Here, λ' is a stand-in for the set of control points λ (line 1). At this point, if the holding agent of λ_m, denoted by A_m, currently holds fewer nodes than the value of $uniformVal$, then λ_m becomes the desired control point for η_i (lines $3 - 4$). Here, the function $p(\lambda_m, A_m)$ represents the current number of nodes held by the agent A_m. If this is not the case, λ_m is excluded from λ', and the process is repeated (lines $6 - 8$). Now, if none of the control points of λ' satisfies the condition of line 3, we assign η_i to its closest control point that does not already associate the most number of non-control point nodes among all the control points λ (lines $9 - 11$). This is important because in this way we can ensure that no agent corresponding to a control point ends up holding too many nodes. Notably, in case of a tie in either or both of the functions in lines 2 and 10, priority should be given to the control point whose associated agent possesses higher computational power. Thus, we can utilize the disparity in agents' computational capabilities (i.e. processing power). Hence, Algorithm 2 returns the control point to line 11 of Algorithm 1, which is denoted by λ_{cp}. Afterwards, line 12 assigns node η_i to the agent holding its associated control point λ_{cp}. As a result, we produce a mapping where a high-degree node is held by the same agent as its connected neighbours in most cases. Such a mapping experiences an additional axiomatic benefit; that is, the intra-agent messages greatly outnumber more expensive inter-agent messages. This is because the majority of the messages generated by the high-degree nodes are transmitted by means of the intra-agent communication.

In the example of Figure 1, the unweighted path costs of the non-control points nodes B, C and E from control point A are one, one and two, respectively. In contrast, the path costs are two, two and one respectively from control point D. According to the regulation of **MNA**, nodes B and C will be associated with control point A, as they have the shortest path from A compared to D. Thus, along with node A, both nodes B and C are eventually held by agent A_1. In the same way, node E picks control point D, and both of them are held by agent A_2. Finally, the mapping obtained by following the process of **MNA** is δ_1 which significantly outperforms δ_2, as already illustrated in the explanation of Figure 2 (see the previous section). The time complexity of the **MNA** algorithm involves two parts. Firstly, $O(k + (N - k) \log k)$ for finding the k-largest nodes (i.e. control points) from N nodes. Secondly, $O(N \log N)$ for choosing suitable control points for the rest of the nodes in G. The overall complexity is therefore $O(N \log N)$ as the value of k is always smaller (or in the worst case, equal) to the number of nodes N.

3.2 Decentralized Version of MNA

Until this point, **MNA** considers those DCOP settings where a node-to-agent mapping is not included as a part of the problem definition, or considerable flexibility exists in choosing the mapping in a centralized manner. However, as discussed in Section 1, the assignment is assumed as a part of the problem in a number of applications, and

as such, the centralized approach is not suitable for them. Moreover, it is important for **MNA** to cope with settings that are not impervious to the introduction of new nodes (and the departure of existing nodes), even after the node-to-agent mapping is done or given. In order to yield the benefits similar to that of the centralized version in such cases, we introduce a decentralized version of **MNA** (i.e. *Steps* $1-4$). To be precise, this particular version of **MNA** can be used before initiating the message passing in applications where the mapping is given a priori; at the same time, it can be used in the event of a change within the graphical representation G during the runtime of a GDL-based DCOP algorithm.

- ***Step* 1: Token Generation.** Each agent $A_j \in A$ generates a token that contains degree $deg(\eta_i)$ and domain info D_i for each node η_i it currently holds. The token also contains $cap(A_j)$, which represents the computation capability (i.e. processing power) of agent A_j.
- ***Step* 2: Multicast Token.** Each agent A_j (or the agents that experience change in G at runtime) shares its token to agents holding nodes within the path distance of length l in G. To be able to ensure that contiguous nodes are being held by the same agent in most cases, it is recommended that the value of l is not too large.[3] Moreover, larger values of l would mean more messages are exchanged, thus eventually increasing overall communication costs.
- ***Step* 3: Request Message.** Based on the information of degree and domain from the received tokens, each agent A_j (or only the receiving agents in the event of change) decides whether it needs to hand over one or more nodes it is holding to some other agent(s). The decision should be taken based on the main feature of **MNA**; that is, an agent should hold the least number of high-degree nodes. Note that, in the case of a tie, priority should be given to an agent that possesses higher processing power (i.e. $cap(A_j)$). Then, each of the deciding agents sends a single unicast request message to each of the agents it wants to relinquish its one or more nodes to.
- ***Step* 4: Response Message.** Finally, considering all the received *Request Messages*, an agent takes a decision (based on the main feature of **MNA** and *cap*()) about each node it received request(s) for. Then, it sends a message in response to each of the *Request Messages*, where the value 1 is used to mark the nodes it is willing to hold, and 0 is used otherwise.

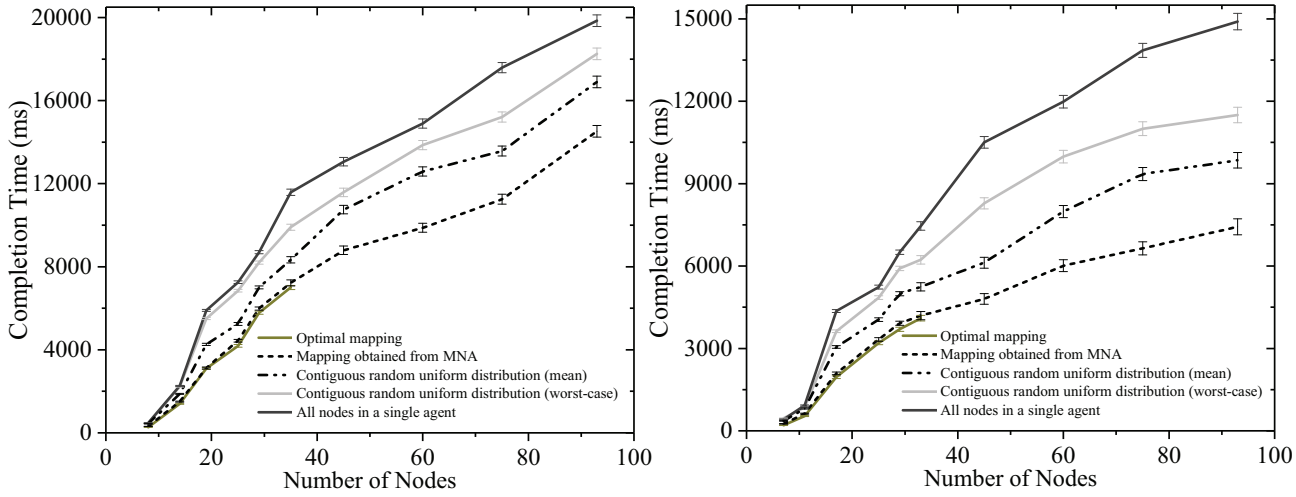
In terms of complexity, concurrently, each agent A_j is observed to generate its own token, which is a small message that contains its nodes' degree, domain information and $cap(A_j)$ based on existing data. Additionally, two decision operations are performed in *Steps* $3-4$ of decentralized **MNA**. Thus, the overall computation complexity is $O(2)$, and in effect, negligible with regard to time. Nevertheless, in *Step* 2, the agent transmits the token (i.e. a small size message) to the holding agents of nodes within the path distance l in G. Since the value of l and the token size is usually small, the overall communication complexity of this approach is linear in terms of time (see Figure 4 and its discussion for empirical evidence).

[3]By considering the value of l within the range 3 to 5, we empirically observe a similar performance between decentralized **MNA** and it's centralized version.

4 EMPIRICAL EVALUATION

We now empirically evaluate the performance of **MNA**[4] in terms of completion time, and compare it with the optimal mapping. As finding an optimal mapping is not feasible for large-scale settings (see Figure 4), we also compare **MNA** with two more benchmarks: (i) a centralized approach, where all the nodes are assumed to be held by a single agent, and (ii) a contiguous random uniform distribution (i.e. mapping). We choose the former as a benchmark to check whether distributing to many agents is indeed necessary. On the other hand, the latter checks the impact of doing this mapping in a simple way, similar to the method used by SECP (see Section 1). All the experiments were performed on a simulator in which we generated different instances of the constraint graph that have a varying number of nodes from 7 to around 100, and the degree of each node is randomly chosen from the range 1 to 7. In the simulation, we made use of the so-called "event-based dependency graph" method (see Section 2 for details) to obtain the completion time for a particular node-to-agent mapping of a constraint graph. In order to accomplish this, we performed an independent set of experiments to generate each node's computation cost (i.e. time) in advance. Here, we consider the domain size of all the nodes in the range of $11-30$. To obtain a node's computation time for all its messages, we initially generated 20 messages of varying sizes from that range, and then averaged the time elapsed in computing the messages. We did so to reflect the growth of search space in the generation of a message by an individual node with an increase in the node's degree and domain size in a DCOP [5, 11, 18]. For example, the computation cost (i.e. the time required to generate each message) of a node with degree 5 is calculated by taking the average duration to compute 20 messages of following complexities: $(11^5, 12^5, \ldots, 30^5)$, where degree $n = 5$ and domain size $d = (11, 12, \ldots, 30)$. While these experiments were performed in a simulator, it is worth mentioning that we use the FRODO repository [23] to generate utility (i.e. cost) tables of such complexities. Meanwhile, we used a network simulator tool (GNS3) in order to obtain the intra-agent and inter-agent communication costs in terms of time [25]. It has been observed that the former type of communication is a few times faster than the latter because we can take the underline network cost into account by using GNS3. Notably, we obtained the values (costs) of the parameters (i.e. computation and communication) through independent empirical observations (and in advance), so as to accurately report the comparative performance from different conceivable mapping approaches without being affected by any application-specific factors (e.g. hazardous communication in disaster response scenarios). Moreover, the exact value of communication cost (in terms of time), which has a significant impact on the overall completion time of a DCOP algorithm, cannot be ascertained accurately in a simulated environment that runs on a single machine (or even a few machines), implying that this would not reflect the application-specific situation such as disaster response, sensor networks, etc. Therefore, we chose to carry out such controlled and systematic experiments wherein the results are neither affected nor generated by skipping several implementation and application-specific issues. Without loss of generality, the comparative results are reported

[4]Note that both centralized and decentralized **MNA** provide comparable node-to-agent mapping, depending on the choice of the value l in the decentralized version.



(a) Random constraint graphs **(b) Scale-free constraint graphs.**

Figure 3: Empirical results for different instances of the constraint graphs with the number of nodes and the number of agents ratio: $(2-12)$**. Error bars are calculated using standard error of the mean.**

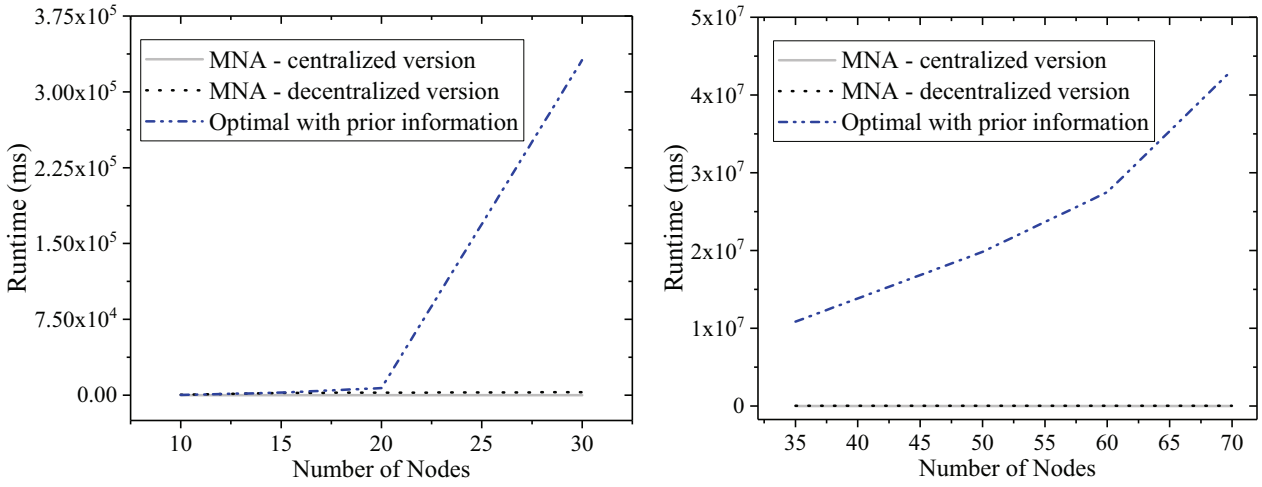
for a single round of message passing[5] for the constraint graphs with cycles, since the results for a single round stand in as a proportional representation for multiple rounds required for the cyclic constraint graphs in this experiment. Nevertheless, we consider the total completion time of the message passing operation for acyclic graphs, as they converge after a single round of message passing. Finally, we report the results averaged over 20 test runs in Figure 3, recording standard errors to ensure statistical significance. Note that the simulator is being implemented and run in an Intel *i*7 Quadcore 3.4GHz machine with 16GB of RAM.

Figure 3a illustrates the comparative results considering the number of nodes-agents ratio from the range $(2-12)$. We found that the results are comparable for settings with higher node-agent ratios. The completion time considering the obtained mapping from **MNA** is compared with the optimal mapping for a particular constraint graph. To report the optimal result for the constraint graph, we run our simulation for all possible uniform mappings. Note that the results depicted in Figure 3 do not include the time required to run the mapping algorithm (**MNA** or optimal) itself, but rather illustrate the completion time of the message passing based on the obtained mapping. We discuss the run-time of the algorithms shortly. In Figure 3a, the dark yellow line indicates the time to complete the message passing from the optimal mappings. As finding such optimal results through this exhaustive approach is not practicable for larger settings, we can only report this up to the constraint graph of 35 nodes (see Figure 4). Here, the dashed black line represents the completion time on the mapping obtained from **MNA**. Significantly, **MNA** always performs at a level of at least 90% of the optimal one. Moreover, in a number of instances, we observe that **MNA** provides the optimal performance. The solid black line represents the outcome from the centralized system, where a single agent holds all the nodes. It performs worse in all the instances because of the fact that an agent cannot compute more than a single message at a time. Even though all the communications are intra-agent in this case, the waiting time for the nodes eventually increases with the growth of the number of nodes. Afterwards, the dashed-dot-dot black line of Figure 3a shows the results of the mean of $10-50$ randomly taken contiguous uniform mappings for each constraint graph. As observed, **MNA** takes around $17-32\%$ less time compared to this benchmark for the constraint graphs having a number of nodes ranging from 7 to 35. Furthermore, we report 16% to around 23% performance gain of **MNA** in the larger constraint graphs compared to the same benchmark. Finally, the solid grey line reports the worst case outcome from the randomly taken contiguous uniform distributions to indicate the possible impact of doing this mapping in a trivial way. In the worst case, a randomly taken uniform distribution performs 25% to around 38% (i.e. around 1.23 to 1.6 times) slower than the mapping obtained from **MNA**.

The same experiments were performed with scale-free graphs [2], and Figure 3b illustrates those results. Although the results are comparable for both types of constraint graphs, we found a notable difference for larger settings. The performances of **MNA** compared to the contiguous random uniform distributions are better (i.e. 24% to around 33% for contiguous random uniform distributions (mean), and 30% to around 43% for the worst case) than what we observed in Figure 3a for the constraint graphs of around 70 nodes or more. This is because the degree distribution of a scale-free graph follows a power law that allows a small subset of nodes to possess much higher degrees than the rest of the nodes in a graph. This phenomenon is particularly suitable for **MNA** to obtain a good node-to-agent mapping.

The aforementioned results clearly show a significant speed-up of message passing algorithms, when they are applied based on the mapping obtained from **MNA**. Nevertheless, we need to ensure that running **MNA** itself is not prohibitively expensive (since it is an additional preprocessing cost on top of the DCOP algorithms).

[5]We report results based on the standard message passing protocol, also known as the message update rule, followed by GDL-based algorithms. See [1, 12] for more details.



(a) Constraint graphs consist of $10-30$ nodes. **(b) Constraint graphs consist of $35-70$ nodes.**

Figure 4: Comparative runtime to obtain the node-to-agent mapping: MNA versus Optimal.

To this end, we need to consider the time **MNA** (centralized and decentralized versions) actually takes to obtain the desired node-to-agent mapping for different constraint graphs, and compare this with the time to obtain the optimal mapping. To report the result for the decentralized version, we consider a processing unit with separate memory of a High Performance Computing (HPC) cluster as an agent. Specifically, Figures 4a and 4b show the results for the constraint graphs with the number of nodes ranging from 10 to 30 and 35 to 70, respectively. It is clear from the grey lines that the centralized version of **MNA** takes linear time to find the mapping for each of the constraint graphs. Although we observe that the decentralized version (dotted black lines) takes slightly more time than its centralized counterpart, it does not incur such delays that would make it prohibitively expensive to deploy. On the other hand, the results illustrated in dashed-dot-dot blue lines show that obtaining the optimal mapping is not practicable or is prohibitively expensive for the constraint graphs of around 25 nodes or more. Note that, to obtain the optimal mapping by considering all contiguous uniform distributions, we had to assume that the computation and communication cost of each messages are known in advance. This is generally unknown prior to executing an optimization algorithm. Taken together, finding an optimal mapping is practically infeasible, while the overall cost of **MNA** is linear.

5 CONCLUSIONS

This paper explores an important gap in the literature, namely the problem of finding good mappings of nodes to agents in DCOPs. As the choice of the assignment can have a significant impact on the completion time of the algorithms, finding good assignments is important. To address this, we propose **MNA**, a heuristic that provides an effective node-to-agent mapping for a DCOP so that the overall completion time of the optimization process can be minimized. To do so, we begin by formulating this specific phase of node-to-agent mapping as an optimization problem in such a manner that **MNA** can be applied to all GDL-based algorithms operating on different graphical representations. Finally, we empirically evaluate the performance of our approach in terms of completion time, showing that it does perform at a level of around $90\% - 100\%$ of the optimal mapping, which is computationally infeasible to obtain in practice. Our results also denote an acceleration of $16\% - 40\%$ as compared to the state-of-the-art, implying that a message passing algorithm can perform $1.2 - 1.7$ times faster when using **MNA**-generated node-to-agent mappings. At the end, we empirically illustrate that the speed-up can be attained with the expense of a linear run-time cost, which is significant given that an optimal mapping is indeed prohibitively expensive. In future work, we intend to investigate whether this algorithm can be applied to other classes (i.e. non-GDL) of DCOP algorithms (e.g. search-based [14, 27] and sampling-based [6, 17]), as well as how much speed-ups can be achieved for them.

ACKNOWLEDGMENTS

We acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of some of the experiments. Long Tran-Thanh is supported by the EPSRC funded project STRICT (EP/N02026X/1). This research is also partially supported by NSF grant 1550662. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or agencies.

REFERENCES

[1] S. M. Aji and R.J. McEliece. 2000. The generalized distributive law. *IEEE Transactions on Information Theory* 46, 2 (2000), 325–343.
[2] A. Barabási, R. Albert, and H. Jeong. 1999. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications* 272, 1 (1999), 173–187.
[3] J. B. Cerquides, A. Farinelli, P. Meseguer, and S. D Ramchurn. 2013. A tutorial on optimization for multi-agent systems. *Computer Journal* 57 (2013), 799–824.
[4] A. Farinelli, A. Rogers, and N. R. Jennings. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous Agents and Multi-Agent Systems* 28, 3 (2014), 337–380.

[5] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Vol. 2. 639–646.

[6] F. Fioretto, F. Campeotto, L. Da Rin Fioretto, W. Yeoh, and E. Pontelli. 2014. GD-GIBBS: a GPU-based sampling algorithm for solving distributed constraint optimization problems. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 1339–1340.

[7] F. Fioretto, E. Pontelli, and W. Yeoh. 2018. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research* 61 (2018), 623–698.

[8] F. Fioretto, W. Yeoh, and E. Pontelli. 2016. Multi-Variable Agent Decomposition for DCOPs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2480–2486.

[9] F Fioretto, W. Yeoh, and E. Pontelli. 2017. A multiagent system approach to scheduling devices in smart homes. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 981–989.

[10] S. Fortune. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2, 1-4 (1987), 153–174.

[11] Y. Kim and V. Lesser. 2013. Improved max-sum algorithm for DCOP with n-ary constraints. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 191–198.

[12] F. R. Kschischang, B. J Frey, and H.A. Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47, 2 (2001), 498–519.

[13] A. R. Leite, F. Enembreck, and J. A. Barthès. 2014. Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications* 41, 11 (2014), 5139–5157.

[14] R. T. Maheswaran, J. P. Pearce, and M. Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems (ISCA PDCS)*. 432–439.

[15] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Vol. 1. 310–317.

[16] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161, 1 (2005), 149–180.

[17] B. Ottens, C. Dimitrakakis, and B. Faltings. 2017. DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. *ACM Transactions on Intelligent Systems and Technology* 8, 5 (2017), 1–27.

[18] A. Petcu and B. Faltings. 2005. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*. 266–271.

[19] A. Rogers, A. Farinelli, R. Stranders, and N.R. Jennings. 2011. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* (2011), 730–759.

[20] P. Rust, G. Picard, and F. Ramparany. 2016. Using Message-Passing DCOP Algorithms to Solve Energy-Efficient Smart Environment Configuration Problems.. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 468–474.

[21] N. Stefanovitch, A. Farinelli, A. Rogers, and N. R. Jennings. 2011. Resource-aware junction trees for efficient multi-agent coordination. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Vol. 1. 363–370.

[22] E. A. Sultanik, R. N. Lass, and W. C. Regli. 2008. DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *Proceedings of the 7th international Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS): Demo Papers*. 1667–1668.

[23] L. Thomas, O. Brammert, and S. Radoslaw. 2009. FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. In *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*. 160–164. https://frodo-ai.tech.

[24] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides. 2009. Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Vol. 2. 1239–1240.

[25] C. Welsh. 2013. *GNS3 network simulation guide*. Packt Publishers.

[26] H. Yedidsion and R. Zivan. 2016. Applying DCOP_MST to a Team of Mobile Robots with Directional Sensing Abilities. In *Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 1357–1358.

[27] W. Yeoh, A. Felner, and S. Koenig. 2010. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal of Artificial Intelligence Research* 38 (2010), 85–133.

[28] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara. 2014. Distributed constraint optimization for teams of mobile sensing agents. *Autonomous Agents and Multi-Agent Systems* 29, 3 (2014), 495–536.