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

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

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

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

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 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, 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.

The agents act (i.e., generate and transmit messages) on behalf of the nodes they hold.
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 Mapping of Nodes to the participating Agents (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 \( (X, D, F, A, \delta) \) [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 \( x_i \in X \) defining the relationship among the variables in \( x_i \). Thus, the function \( F_i(x_i) \) denotes the value for each possible assignment of the variables in \( x_i \). 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, \( \eta \) stands for the set of nodes within the constraint graph \( G \). As a result of the mapping represented by \( \delta \), 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).

\[
P(A) = \bigcup_{j=1}^{k} G_j \quad \forall j' \neq j : (G_j \cap G_{j'}) = \emptyset
\]  

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, \( X^* = \arg \max_X \sum_{i=1}^{n} F_i(x_i) \). 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 \( E_G(A, P) \), where \( A \) 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 \( E_G(A, P) \). 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 \( T(A, P) \) 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 \( E_j \).

\[
T(A, P) = \max_{\forall E_i, E_j \in E_G(A, P)} v(E_i, E_j)
\]  

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

\(^2\)In a multi-processing capable setting, each processing unit with separate memory can be considered as an agent.
The complete process of MNA’s centralized version is detailed in Algorithm 1. As aforementioned, it aims to reach a point where no
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 of a DCOP that is going to be solved by deploying a GDL-based algorithm $A$. Here, $G$ consists of a set $\eta = \{\eta_1, \eta_2, \ldots, \eta_N\}$ of $N$ nodes and a set $A = \{A_1, A_2, \ldots, A_k\}$ of $k$ agents. At the end, Algorithm 1 returns $\delta : \eta \rightarrow A$, that is the mapping $\delta$ of the nodes $\eta$ 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 $deg = \{\text{deg}(\eta_1), \text{deg}(\eta_2), \ldots, \text{deg}(\eta_N)\}$ be the set where each $\text{deg}(\eta_i) \in deg$ stands for the degree/number of connected nodes of $\eta_i$.
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 $\eta_i$ has to get its preferred value.
3. $\text{uniformVal} \leftarrow N/k$
4. if ($\text{deg}(\eta_1) - \text{deg}(\eta_k) == 0) \land (|D_1| - |D_k|) == 0$, where $\forall \eta_i, \eta_j \in \eta, \forall D_i, D_j \in D$ then \hspace{1cm} // Contiguous uniform node-to-agent mapping, when the nodes possess similar degree and equal domain size.
5. else
6. $\lambda \leftarrow k\text{-largestNodes}(G, \eta, k)$ \hspace{1cm} // Find a set $\lambda$ of $k$ largest nodes from $\eta$ in terms of degree.
7. $\delta : \lambda \rightarrow A$ \hspace{1cm} // Distribute the nodes of $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ to $A$ such that each agent holds a single node.
8. $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ are the control points of the graph $G$.
9. foreach node $\eta_i \in \eta \setminus \lambda$ do
10. $\lambda_{cp} \leftarrow \text{minDistance}(G, \eta_i, \lambda, \text{uniformVal})$, where $\lambda_{cp} \in \lambda$ \hspace{1cm} // Call Algorithm 2: choose the suitable control point $\lambda_{cp}$ for the node $\eta_i$.
11. $\delta : \eta_i \rightarrow \lambda_{cp}$ \hspace{1cm} // allocate $\eta_i$ to the agent $A_{\lambda_{cp}}$ that holds the control point $\lambda_{cp}$.
12. return $\delta : \eta \rightarrow A$.

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 $\eta$ 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 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 $\text{uniformVal}$, which is the ratio of the number of nodes $N$ and the number of agents $k$ in $G$. 

Algorithm 1: MNA ($G, \eta, A, k$)

**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 $A = \{A_1, A_2, \ldots, A_k\}$ is the set of $k$ agents participating in the optimization process, where $k \leq N$. $A$ stands for the deployed GDL-based DCOP algorithm.

**Output:** Mapping $\delta$ of the nodes of $\eta$ to their associated agents $A$ (i.e. $\delta : \eta \rightarrow A$), based algorithm $A$ deployed algorithm $A$.
Algorithm 2: minDistance(G, \eta_i, \lambda, uniformVal)

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

Output: \lambda_m \in \lambda, the corresponding control point for \eta_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 \lambda_m holds fewer nodes than the value of uniformVal.
   4. return \lambda_m
5. else
6. \lambda' \leftarrow \lambda' \setminus \lambda_m
7. if \lambda' \neq \emptyset then go to line 2
8. \lambda_m \leftarrow all_sPath(G, \eta_i, \lambda) // assign \eta_i to the closest control point that does not currently associates the most number of non-control point nodes among \lambda.
9. return \lambda_m


example of Figure 1, the agents A_1 and A_2 are participating in the optimization process, hence the value of \kappa 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 possess 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 \text{minDistance}(G, \eta_i, \lambda, uniformVal), detailed in the pseudo-code of Algorithm 2, takes as input a non-control point node \eta_i, the subset \lambda of \eta that acts as the control points and previously computed uniformVal, and then finds a suitable control point \lambda_{cp} \in \lambda for \eta_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 \eta_i that possesses the shortest path from \eta_i within the constraint graph G, and G is considered as an unweighted graph during this process. Here, \lambda' is a stand-in for the set of control points \lambda (line 1). At this point, if the holding agent of \lambda_m, denoted by A_m, currently holds fewer nodes than the value of uniformVal, then \lambda_m becomes the desired control point for \eta_i (lines 3 – 4).

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 \delta_1 which significantly outperforms \delta_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 $\text{deg}(\eta_i)$ and domain info $D_l$ for each node $\eta_i$ it currently holds. The token also contains $\text{cap}(A_j)$, 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.\footnote{By considering the value of $l$ within the range 3 to 5, we empirically observe a similar performance between decentralized MNA and its centralized version.} 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. $\text{cap}(A_j)$). 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 $\text{cap}(\cdot)$) 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 $\text{cap}(A_j)$ 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).

### 4 Empirical Evaluation

We now empirically evaluate the performance of MNA\footnote{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.} 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 (GNSS3) 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 GNSS3. 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.
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 i7 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.

---

**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.
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

ordination 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.

GD-GIBBS: a GPU-based sampling algorithm for solving distributed constraint
optimization problems. In Proceedings of the 13th International Conference on

procedures and applications: A survey. Journal of Artificial Intelligence Research

2480–2486.

scheduling devices in smart homes. In Proceedings of the 16th Conference on


constraints. In Proceedings of the 12th International Conference on Autonomous

498–519.

mization problems: Review and perspectives. Expert Systems with Applications

for DCOP: A Graphical-Game-Based Approach. In Proceedings of the ISCA 17th
International Conference on Parallel and Distributed Computing Systems (ISCA

2004. Taking DCOP to the real world: Efficient complete solutions for distributed
multi-event scheduling. In Proceedings of the 3rd International Joint Conference on

distributed constraint optimization with quality guarantees. Artificial Intelligence

Bound Approach to Distributed Constraint Optimization Problems. ACM Trans-
rations on Intelligent Systems and Technology 8, 5 (2017), 1–27.

optimization. In Proceedings of the 19th International Joint Conference on Arti-
ficial Intelligence (IJCAI). 266–271.

mate decentralised coordination via the max-sum algorithm. Artificial Intelligence

rithms to Solve Energy-Efficient Smart Environment Configuration Problems.
In Proceedings of the 25th International Joint Conference on Artificial Intelligence
(IJCAI). 468–474.

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.

lating and deploying distributed constraint reasoning algorithms. In Proceedings
of the 7th International Joint Conference on Autonomous Agents and Multi-Agent

Framework for Distributed Constraint Optimization. In Proceedings of the IJCAI'09
tech.

Action-GDL, a new complete algorithm for DCOPs. In Proceedings of the 6th In-
ternational Conference on Autonomous Agents and Multi-Agent Systems (AAMAS),
Vol. 2. 1239–1240.


Robots with Directional Sensing Abilities. In Proceedings of the 15th Interna-
tional Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).
1667–1668.

and-Bound DCOP Algorithm. In Proceedings of the 13th International Conference on

constraint optimization for teams of mobile sensing agents. Autonomous Agents