2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Distributed constraint optimization for continuous mobile sensor coordination

Jeroen Fransman[a], Joris Sijs[a], Henry Dol[b], Erik Theunissen[c], Bart De Schutter[a]

***Abstract*— DCOP (Distributed Constraint Optimization Problem) is a framework for representing distributed multi-agent problems. However, it only allows discrete values for the decision variables, which limits its application for real-world problems. In this paper, an extension of DCOP is investigated to handle variables with continuous domains. Additionally, an iterative any-time algorithm Compression-DPOP (C-DPOP) is presented that is based on the Distributed Pseudo-tree Optimization Procedure (DPOP). C-DPOP iteratively samples the search space in order to handle problems that are restricted by time and memory limitations. The performance of the algorithm is examined through a mobile sensor coordination problem. The proposed algorithm outperforms DPOP with uniform sampling regarding both resource requirement and performance.**

I. INTRODUCTION

A wide range of real-world problems can be modelled as a Multi-Agent System (MAS): scheduling problems [1], mobile sensor coordination [2], hierarchical task networks mapping [3], and control of modern robotics such as RoboCup Rescue [4]. Real-world problems often involve agents that are bound by inter-agent constraints (communication, movement) and by limited resources (memory, processing power). Likewise, bounded computation time is important to ensure safety when operating in a (uncertain) dynamic environment, for example, obstacles can only be avoided when the reaction is executed in a timely manner. The main challenge of MAS is to coordinate the actions of the agents by a distributed process, since a centralized process would become intractable for large number of agents.

The Distributed Constraint Optimization Problem (DCOP) framework has been introduced to represent problems that are naturally distributed [5]. A DCOP is typically represented as a constraint graph, where nodes represent the variables of the problem, and edges represent a constraint or utility relation between the variables. The agents coordinate their actions by exchanging messages about the utility of their interactions. The utility values represent the differences in cost and benefits of the actions for the individual agents. By using utility, individual goals and internal dynamics are abstracted and hidden from other agents, which makes modelling of real-world problems less complex, since not all interactions need to be modeled in great detail. This results in modelling versatility and simplicity.

DCOP defines control variables with finite discrete domains, which limits its use for problems with continuous variables and utility functions. The latter type of problems are less studied within the DCOP framework [4]. For continuous path planning and multi-robot coordination/collision avoidance specific solutions exist. In the work of [6], a set of rapidly-exploring random trees is used as discrete domain for the trajectories. An alternative approach is presented in [7], where the utility functions are approximated as piecewise linear functions. These methods assume that the utility tables are readily available or can be computed beforehand. In real-world situations this is often not the case, and calculation of these values could be subject to large computational requirements. A generic solution would be to transform the continuous domains into discrete domains by sampling up to a desired resolution. However, this would result in a rapid growth in the computational complexity. The complexity growth is due to the exponentially growth of the *size* of the search space with respect to the interconnectivity and the size of the domains of the variables.

Numerous solvers for DCOP have been proposed; for a detailed overview of the taxonomy of DCOP algorithms, the reader is referred to [8], [9]. In this paper, the focus is on the DPOP algorithm, since it requires a fixed number of communication steps. This property makes it suitable for real-world problems, since it bounds the interactions between the agents. DPOP uses dynamic programming elements to communicate accumulated information among agents, such that every new message enables a more informed selection of the optimal control variables. To reduce the complexity growth DPOP has been extended in numerous manners. These approaches can be divided into complete and approximate solvers. Complete solvers are guaranteed to find the (global) optimal solution, approximate solvers do not. Optimally solving DPOP is NP-HARD, which makes providing real-time guarantees unattainable [10]. Therefore, approximate solutions are considered to be a valid option for a problem with limited resources.

In local optimal solutions the computational requirements are reduced in three predominant methods:

1) *Reducing the number of interconnections* by iteratively adding interactions and re-evaluating the problem (I-DCOP [1]) until an acceptable solution is found. This reduces the maximum size of the message, but could lead to infeasible solutions when constraints of the problem are neglected.

[a] Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, j.e.fransman@tudelft.nl
[b] Netherlands Organisation for Applied Scientific Research (TNO), The Hague, The Netherlands
[c] Netherlands Defence Academy (NLDA), Den Helder, The Netherlands

2) *Restricting the number of variables in the messages* by dropping a set of variables from the message when a maximum message size is violated. This trades solution quality against computational complexity (A-DPOP [11]) based on lower and upper utility bounds. For memory restricted agents, this could lead to arbitrarily poor performance.
3) *Limit the growth of the messages*, by filtering inferior solutions based on global lower and upper utility bounds. This method is used to improve DPOP (MB-DPOP [12]), in combination with the Generalized Distributive Law [4] or max-sum algorithms [13]. This method can achieve a major reduction in communication; however this cannot be guaranteed. The benefits will depend greatly on the specific problem at hand.

In the current paper a fourth option is explored:

4) *Regulating the search space by confining the number of samples of the continuous domains* and iteratively applying DPOP while contracting the continuous domains around intermediate (local) solutions. Consequently, the search space can be sampled more effectively based on local optima without the need for larger computational resources.

The remainder of this paper is organized as follows. First, Section II defines the DCOP model and its extension towards continuous domains. Then, Section III elaborates on the DPOP algorithm, which is extended in Section IV by the proposed algorithm. Next, Section V defines the continuous mobile sensor coordination problem that is used to compare the performance and memory and computation time requirements of DPOP and the proposed algorithm. The implementation of the mobile sensor coordination problem is detailed and the results are analysed. Finally, Section VI summarizes the results and defines the future work.

II. DCOP DEFINITIONS & NOTATION

A Distributed Constraint Optimization Problem (DCOP) is defined by a tuple $< A, X, D, F, G >$ [4], where:

A	is a set of agents, i.e. $A = \{a_1, a_2, \ldots, a_{n_A}\}$ and $n_A \in \mathbb{N}$ is the number of agents.
X	is a set of decision variables, i.e. $X = \{X_i \mid \forall a_i \in A\}$. $X_i = \{x_{i,1}, \ldots, x_{i,n_{X_i}}\}$ is the set of variables of agent a_i, and $n_{X_i} \in \mathbb{N}$ is its number of variables. The number of elements in set X_i is also denotes as $\lvert X_i \rvert$.
D	is a set of all domains of all variables, i.e. $D = \{D_p \mid \forall x_p \in X\}$ and the domain of variable x_p is defined as $D_p = \{d_{p,1}, d_{p,2}, \ldots, d_{p,n_{D_p}}\}$, where $n_{D_p} \in \mathbb{N}$ defines the number of elements within in the domain D_p.
F	is a set of utility functions, i.e. $F = \{f_k : S(f_k) \rightarrow \mathbb{R} \cup \{-\infty\}\}$. Hard constraints are modelled as $-\infty$ utility. Each function $f_k \in F$ is defined over a subset $S(f_k) \in X$, also referred to as the scope of the function. The scope of F_i is similarly defined as $S(F_i) = \{S(f_k) \lvert \forall f_k \in F_i\}$. Every agent a_i knows the utility functions that involve its own decision variables X_i. Formally, agent a_i knows the subset $F_i \subseteq F$, $F_i = \{f_k \in F \mid \exists f_k$ such that $S(f_k) \cap X_i \neq \emptyset\}$.
G	the global objective function that captures the aggregated utility of a complete allocation of all variables, denoted as $\mathfrak{X}$. An *allocation* $\mathfrak{X}$ maps each variable $x \in X$ to a value in its domain D, $\mathfrak{X} : X \rightarrow D$. The goal is to find an optimal allocation $\mathfrak{X}^* = \arg\max_{\mathfrak{X}} G(\mathfrak{X})$, where $G(\mathfrak{X}) = \sum_{f_k \in F} f_k(\mathfrak{X}[S(f_k)])$. The *projection* of an allocation $\mathfrak{X}$ over a set of variables $X_p \subseteq X$, written $\mathfrak{X}[X_p]$, is a new allocation $\mathfrak{X}_\mathrm{p}$ formed by the values that $\mathfrak{X}$ assigns to the variables in X_p. For example, with $f_1(x_1) = {x_1}^2$, $S(f_1) = \{x_1\}$, $D_1 = \{1, 2\}$, and $\mathfrak{X}_1 = \{x_1 = 2\}$, then $f_1(\mathfrak{X}_1[S(f_1)]) = f_1(\mathfrak{X}_1[x_1]) = f_1(2) = 4$. The *search space* (all possible allocations of the variables) $\mathbf{X} = \prod_{x_p \in X} D_p$ defines each combination of all elements in the domain for the variables in set X, where $\prod$ is the set Cartesian product.

A DCOP is distributed in the sense that agents only interact through variables coupled by a utility function. The function set that is shared by agents a_i and a_j is denoted as $F_{i,j} = \{f_k \mid f_k \in F_i \cap F_j\}$. Likewise, the set of functions of agent a_i that are not shared by other agents is denoted as $F_{-i} = \{f_k \mid S(f_k) \cap S(f_j) = \emptyset, \ \forall \ f_j \in F \setminus F_i\}$.

A. THE C-DCOP MODEL

The Continuous DCOP (C-DCOP) model extends the DCOP model towards variables with continuous domains. Formally, it is a tuple $< A, X, D, F, G >$, where:

A, F, G	are equal to their definition in DCOP.
X	The variable set $X = (X^d, X^c)$, where all variables with a discrete domain x^d belong to set $X^d = \{x_p \mid p = 1, 2, \ldots, n_{x_d}\}$ and all variables with a continuous domain x^c belong to set $X^c = \{x_q \mid q = 1, 2, \ldots, n_{x_c}\}$. The number of discrete variables and continuous variables is denoted as $n_{x^d} \in \mathbb{N}$ and $n_{x^c} \in \mathbb{N}$, respectively.
D	The domain set $D = (D^d, D^c)$, where $D^d = \{D_p^d \mid \forall x_p \in X^d\}$ and $D^c = \{D_q^c \mid \forall x_q \in X^c\}$. The continuous domain of variable x_q is defined by its lower and upper bound as $D_q^c = (\underline{d}_q^c, \bar{d}_q^c)$ indicating the domain over which the variable x_q can take a value.

III. DISTRIBUTED PSEUDO TREE OPTIMIZATION PROCEDURE (DPOP)

Distributed Pseudo-tree Optimization Procedure (DPOP) is a solver for the DCOP framework. DPOP operates over a pseudo tree [14], which is a rooted directed spanning tree, where connected nodes fall in the same branch. Every edge of the pseudo tree is represented by a parent/child (direct) relation or by a backedge for an pseudo parent/pseudo child (indirect) relation. This representation allows for separating the main problem into sub-problems (between branches) and solving these independently before merging into the global assignment [15]. In order to create a pseudo tree from a constraint graph a depth-first search traversal can be executed. In this paper, the Distributed Depth-First-Search (DFS) algorithm [16] is used, since it performs traversals along backedges in parallel, thereby reducing the time complexity. A DFS is defined by assigning the following properties to every node/agent a_i in the tree: the descendant/ancestor agents that are directly connected through a tree edge are indicated by C_i, P_i, respectively. descendant/ancestor agents, that are indirectly connected through a backedge are indicated by PC_i, PP_i, respectively. The set of all connected agents to agent a_i or to its descendants excluding the agent a_i itself is defined as $\mathrm{J}_i = \{\cup_{a_j \in \mathrm{C}_i} \mathrm{J}_j \cup \mathrm{P}_i \cup \mathrm{PP}_i\} \setminus \{a_i\}$.

The DPOP algorithm [5] solves DCOP in three phases:

1) *Pseudo tree construction:* The agents distributively create the pseudo tree structure, by a distributed pseudo tree construction algorithm. Each agent a_i labels all its neighbours as either parent, pseudo parent, child, or pseudo child $(\mathrm{P}_i, \mathrm{PP}_i, \mathrm{C}_i, \mathrm{PC}_i, \mathrm{J}_i)$.
2) *Bottom-up utility propagation:* From the leaves (nodes without children) of the pseudo tree the agents pass a utility message U towards their parents.
   A utility message U^i_j is send by agent a_j to agent a_i based on the shared utility function set $F_{i,j} \subseteq F$, defined as $U^i_j \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{k,p}}$, where $n_{k,p} \in \mathbb{N}$ is the number of elements in D_p of variable $x_p \subset S(F_{i,j})$. In words, it is a multidimensional matrix with one dimension for each variable x_p within $S(F_{i,j})$. Parents receive utility matrices U^i_j from all their children indicating the combined utility for coupled variables between the agents.
   All child matrices are combined as $U^i_{\mathrm{C}_i} = \bigoplus_{a_j \in \mathrm{C}_i} U^i_j$, where the messages are combined by a join operator $\oplus$ as $U^i_{1,2} = U^i_1 \oplus U^i_2$ such that the scope is the union of both scopes, i.e. $S(U^i_{1,2}) = S(U^i_1) \cup S(U^i_2)$. The value of the elements of $U^i_{1,2}$ is the sum of the elements of U^i_1 and U^i_2 for all combinations of $S(U^i_1)$ and $S(U^i_2)$. For example, $U^i_1 = \begin{bmatrix} 1 & 3 \end{bmatrix}$, with $S(U^i_1) = \{x_1\}$ and $U^i_2 = \begin{bmatrix} 3 & 4 \end{bmatrix}$, with $S(U^i_2) = \{x_2\}$. Combining these matrices results in $U^i_{1,2} = U^i_1 \oplus U^i_2 = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$ with, $S(U^i_{1,2}) = \{x_1, x_2\}$.
   The utility matrices of the children are combined with the local utility matrix $U(F_{-i})$ and the utility matrices of the (pseudo) parents of the agent to form the total utility matrix U^i of agent a_i as
   $U^i = U^i_{\mathrm{C}_i} \oplus U(F_{-i}) \oplus \left(\bigoplus_{a_j \in \{\mathrm{P}_i \cup \mathrm{PP}_i\}} U^j_i \right)$.
   The resulting matrix U^i is optimized over the local variables of the agent a_i. This operation is defined by a projection operator $\perp$ and assigns the maximal utility of allocation of the local variable set X_i. The result is a matrix of lesser cardinality based on X_i, in words, for all values of $\mathbf{X}_i$ the optimal value is chosen with respect to the allocation $\mathfrak{X}_i$.
   Formally, $U^i \perp X_i = \max_{\mathfrak{X}_i \in \mathbf{X}_i} U^i[\mathfrak{X}_i]$, and the scope $S(U^i \perp X_i) = S(U^i) \setminus X_i$. For example, when $U^i = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$ with $S(U^i) = \{x_1, x_2\}$, the resulting projection is $U^i \perp \{x_1\} = \begin{bmatrix} 6 & 7 \end{bmatrix}$ and $\mathfrak{X}_1 = \{\{x_1 = 2 | x_2 = 1\}, \{x_1 = 2 | x_2 = 2\}\}$.
   After the projection, the matrix is sent towards to its parent, $U^{\mathrm{P}_i}_i = U^i \perp X_i$. After the root agent of the pseudo tree (agent without parent) has finished this procedure, the value propagation phase is initiated.
3) *Top-down value propagation:* The root agent has accumulated the combined utility values U^i and is able to choose the optimal assignment of its local variable set X_i, $\mathfrak{X}^*_i = \arg\max_{\mathfrak{X}_i}(U^i \perp X_i)$. The allocation of these values is sent to all the children of the agent. Based on these values the children allocate their own variables, $\mathfrak{X}^*_j = \arg\max_{\mathfrak{X}_j}(U^j \perp \mathfrak{X}^*_i \perp X_j)$.
   Every agents repeats this process until the leaves of the tree are reached, completing the assignment $\mathfrak{X}^*$.

A graphic representation of the DPOP algorithm can be seen in Figure 1, where a simple problem is shown during execution of the algorithm.

IV. Compression DPOP (C-DPOP)

In order to efficiently sample the continuous domains of the variables within the C-DCOP framework, the Compression-DPOP (C-DPOP) algorithm is proposed.
C-DPOP is an approximate any-time iterative algorithm that extends DPOP by dynamically sampling continuous domains by a fixed number of samples per variable, $n_{s_q} \in \mathbb{N}$ for all $x_q \in X^c$. After every iteration, the width of the domain of variable x_q, w_q is decreased according to a compression factor $(0 < c_q < 1)$, which results in an increased domain resolution r_q (sample distance). This process can be stopped at any time if the allowed computation time has expired.

C-DPOP consists of five phases which are iteratively executed by all agents:

1) *Domain sampling:* All continuous domains of agent a_i are sampled based on its upper and lower bound $(\bar{d}_q, \underline{d}_q)$ so to generate discrete domains. Based on the number of samples per domain n_{s_q}, the domain is uniformly sampled $D^c_q \rightarrow D^d_q$ such that,
   $D^d_q = \{d_{q,1}, \ldots, d_{q,n_{s_q}}\}$ and $d_{q,r} = \underline{d}_q + (r-1)\frac{w_q}{n_{s_q}-1}$.
2) *Utility tables creation:* Based on the search space of the discrete domains the utility functions are sampled to calculate the utility values. Every agent calculates the local utilities $U(F_{-i})$, and the utilities of the (pseudo) parents $U^j_i \forall j \in \mathrm{P}_i \cup \mathrm{PP}_i$.
3) *Bottom-up utility propagation:* After the utility tables are generated, the leaves of the tree start by sending their utility matrices to their parents. These matrices are combined with the local utility matrices, which are then projected over the local variables before being sent to the parents. This phase is identical to DPOP.
4) *Top-down value propagation:* The root of the tree initiates the value propagation, after which all children allocate their local variables before sending it to their children. This phase is identical to DPOP.

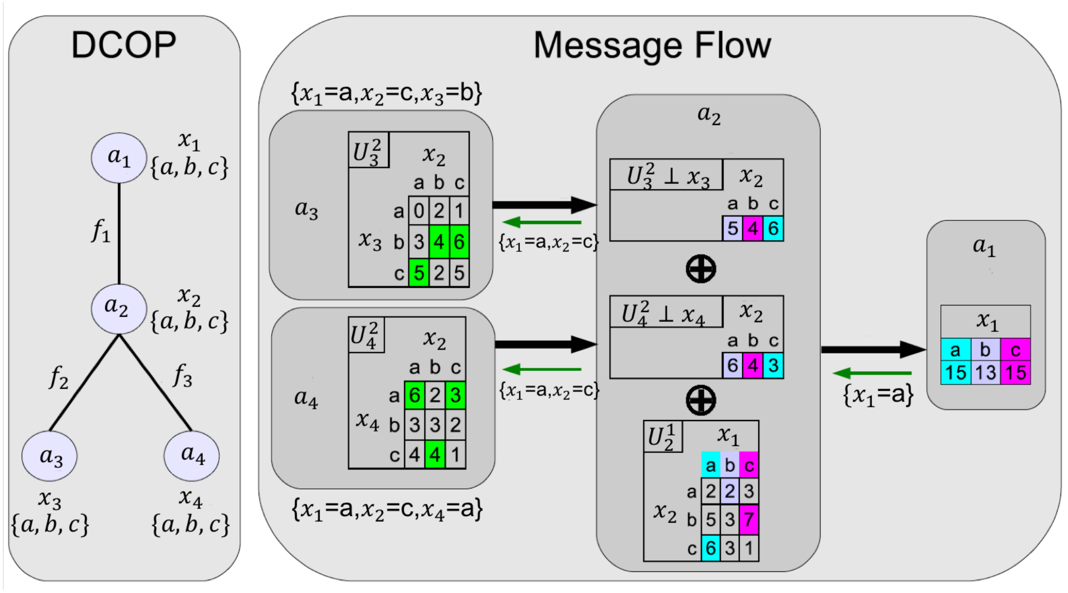


Fig. 1. A simple DCOP problem, adapted from [10] Left: a tree representation where the agents $A = \{a_1, a_2, a_3, a_4\}$ are represented as nodes. The edges represent the utility relations between the variables of the agents $F = \{f_1, f_2, f_3\}$. The variables $X = \{X_1, X_2, X_3, X_4\}$, where $X_i = \{x_i\}$ for $i = 1, 2, 3, 4$. All variables have identical domains $D = \{D_1, D_2, D_3, D_4\}$, where $D_1 = D_2 = D_3 = D_4 = \{a, b, c\}$. Right: the message flow starts from the leaves of the tree (a_3, a_4) based on the projection of their utility matrices over their decision variables. These matrices are combined by agent a_2 by the join operation $(\oplus)$ before projecting out x_2 and sending the result to agent a_1. Afterwards, agent a_1 calculates the optimal assignment for x_1 and sends it to its child (a_2), which assigns the optimal value for x_2 before sending the combined assignment to its children $(a_3$ and $a_4)$. Lastly, agents a_3 and a_4 assign their local values.

5) *Domain compression:* After a (local) optimal allocation for the local variables $\mathfrak{X}_i^*$ is found, the upper and lower bounds of the domains are updated. This is done by compressing the width of the domain $w_q^+ = w_q c_q$, where $w_q == \bar{d}_q - \underline{d}_q$ and centering the domain around the allocated values $\mathfrak{X}_i^*[x_q]$. The new domain is defined as $D_q^c = (\underline{d}_q^c, \bar{d}_q^c)$, $\underline{d}_q^c = \mathfrak{X}_i^*[x_q] - \frac{w_q^+}{2}$, and $\bar{d}_q^c = \mathfrak{X}_i^*[x_q] + \frac{w_q^+}{2}$.

The compression of the domain is a key-part of the algorithm, since it iteratively refines the solution. The proposed restriction strategy is based on contracting grid search. By this, the sampling of the continuous domain is focused around the (local) optimal value (an area with high utility), while the iteratively compression of the domain reduces the exploration (of the area around the found optimum). An overview of the strategy for a one-dimensional domain can be seen in Figure 2.

For resource constraint problems, C-DPOP can be used with a limited number of iterations. The maximum number of iterations is calculated explicitly based on the required number of memory and computation time. The memory requirement is a function of the number of samples used per variable. The total required memory by agent a_i can be calculated as $m_i = |\mathbf{X}_i| m_1$, where $|\mathbf{X}_i|$ is the size of the search space for all variables in X_i, and m_1 is the required number of bytes to store the utility of a single assignment (typically 8 bytes for a float). The required time can be calculated based on the number and execution time of all function evaluations. The number of evaluations is equal to the size of the search space of all variables in the scope of

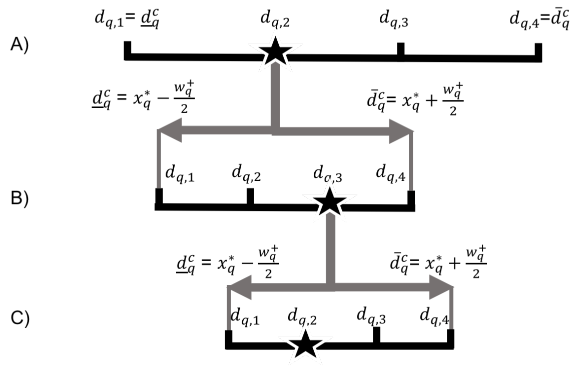
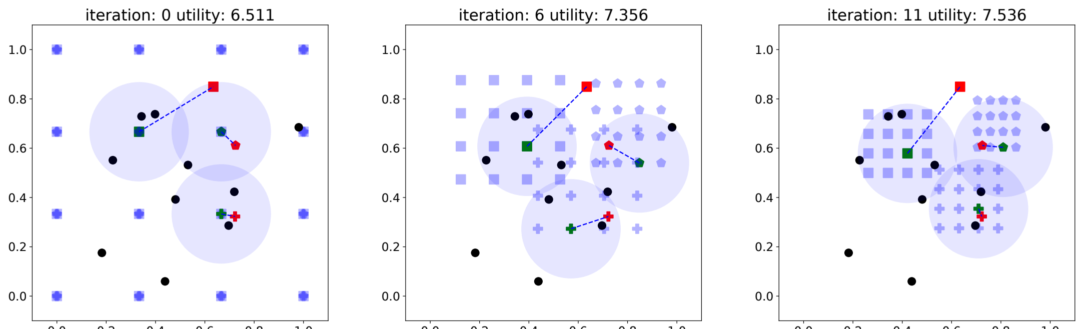

A)
B)
C)
$d_{q,1} = \underline{d}_q^c$
$d_{q,2}$
$d_{q,3}$
$d_{q,4} = \bar{d}_q^c$
$\underline{d}_q^c = x_q^* - \frac{w_q^+}{2}$
$\bar{d}_q^c = x_q^* + \frac{w_q^+}{2}$
$d_{q,1}$
$d_{q,2}$
$d_{q,3}$
$d_{q,4}$

Fig. 2. An overview of the domain compression strategy. Domain D_q^c is updated over three iterations (A, B, C) by reducing the width of the domain $w_q^+ = w_q c_q$ and centering around the optimal value represented as vertical arrows. The optimal value is indicated by a star. The domain is represented as a horizontal black line on which the samples $\{d_{q,r}\}_{r=1}^4$ are shown.

the function, $n_{e_{f_k}} = |\mathbf{X}_{f_k}|$. It is a property of the number of variables in the scope and the size of their domains, since $\mathbf{X}_{f_k} = \prod_{x_q \in S(f_k)} D_q$. The evaluation time of a function f_k is a property of that function and is denoted as t_{f_k}. This property is assumed to be known for all functions $f_k \in F$. Based on these two properties the required computation time for agent a_i can be defined as $t_i = \sum t_{f_k} n_{e_{f_k}}$ for all $f_k \in F_{-i} \cup F_{i,j} \forall j \in \mathrm{P}_i \cup \mathrm{PP}_i$. The achievable resolution r_q^i of agent a_i for variable x_q can be calculated based on the chosen compression factor c_q, the initial width of the domain w_q, the number of samples n_{s_q}, and the number of iterations $n_{I_q} \in \mathbb{N}$, as $r_q^a = c_q^{n_{I_q}-1} \frac{w_q}{n_{s_q}-1}$.



(a) Initial domain sampling. (b) Intermediate domain sampling. (c) Final domain sampling.

Fig. 3. Selecting and compressing of the domain during C-DPOP for a three agent, ten target example. The initial positions of the agents are shown as red markers, their search space (domains) as blue markers, and their optimal value as green markers. The markers are type coded in order to identify the different agents (square, plus, pentagon). The targets are indicated as black circles. Snapshots from three iterations are shown. In (a) the domains can be seen to overlap, seemingly only showing the search space of a single agent. In (b) the domains of the agents are compressed and centred around their found (local) optima. In (c) the final solution is shown with the sensor ranges of the agents shown as transparent blue circles.

V. SENSOR COORDINATION PROBLEM

A. PROBLEM STATEMENT

A mobile sensor coordination problem is used to evaluate the performance and computational requirements of the C-DPOP algorithm. The problem is adapted from [17] where a team of mobile sensors with limited sensing range need to coordinate their positions (from initial locations) in order to minimize the amount of unobserved targets in an area. The problem is extended for agents with limited memory and computation time, and the agents are able to select positions over continuous domains within a two-dimensional plane.

The problem is described within the C-DCOP framework where the position of agent a_i is defined as $X_i = \{x_i, y_i\}$. The available resources are defined as $t_i^{\max}$ for time and $m_i^{\max}$ for memory of agent a_i. The utility functions of the problem are defined as $F = \{F_T, F_A\}$. $F_T = \{f_{t,l}\}_{l=1}^{n_T}$ is the utility function set for sensing the targets, where $n_T \in \mathbb{N}$ is the number of targets. A target T_l is defined as a point (x_l, y_l). The utility function of target l is described as

$$f_{t,l} = \begin{cases} 1 & \text{if } d(T_l, X_j) \leq s_j \\ 0 & \text{if } d(T_l, X_j) > s_j \end{cases},$$

where s_j is the sensing distance of the closest agent (a_j), and $d(T_l, X_j)$ is the Euclidean distance between the location of the target and the position of the agent. The utility function set for the movement is represented as $F_A = \{f_{a,i}\}_{i=1}^{n_A}$, where $f_{a,i} = -d(X_i, I_i)$ and $I_i = (x_i^0, y_i^0)$ is the initial location of agent a_i. Goal function that is the difference between the utility gain from targets in sensor range subtracted with the cost of moving to a new location for all agents, i.e. $G(\mathfrak{X}) = \sum_{f_k \in F} f_k(\mathfrak{X}[S(f_k)])$

B. IMPLEMENTATION

The performance of the proposed C-DPOP algorithm is compared to DPOP with uniform sampling of the domains by considering the achieved utility. The DPOP method samples the continuous domains only once, while C-DPOP iteratively refines the domains. An example of the operation of C-DPOP can be seen in Figure 3. The two methods are compared for randomly generated mobile sensor coordination problems. In every problem the initial location of the agents I and location of the targets T is randomly chosen. The agents can choose any position within this area and therefore the domains of the variables are all equal, $D_{x,i}^c = D_{y,i}^c = (0, 1)$ for all $a_i \in A$. Since all variables have equal domains and all agents have similar variables, the agent subscript and variable subscripts will be neglected for brevity. All agents are assumed to have an equal amount of available memory and computation time. Based on these resources, the maximum achievable number of samples will be limited by either the memory or the computation time. For DPOP, the maximum amount of samples achievable within the available resources are used.

The C-DPOP algorithm requires a fixed number of samples and a compression factor in order to calculate the allowed number of iterations. The number of samples and the compression factor can selected based on properties of the underlying problem. In the case of mobile sensor coordination, the number of samples can be selected to correspond to a minimal allowed resolution. In this case, it is defined with relation to the sensor range ($s_i = 0.2$) in order to achieve overlap within the sensed areas. Based on the area size, the number of samples for C-DPOP is set at 5. The compression factor can be chosen in correspondence with the non-linearity of the underlying problem. A compression factor close to 0 will compress the domain relatively fast, excluding large segments of the domain at every iteration. Thereby, converging to a (local) optimum rapidly, which is not preferable for (highly) non-linear problems since this can exclude segments that hold the global optimum. A compression factor close to 1 will exclude small segments, therefore being able to escape a (local) optimum more easily when a new (local) optimum is found. For the problem at hand the compression factor $c = 0.9$ is chosen.

A comparison of achieved utility for a three agent, ten target problem is shown in Figure 4. During evaluation of the performance of DPOP and C-DPOP, it was found that the utility was highly dependent on the achievable resolution of the algorithms. This difference can be attributed to accurate

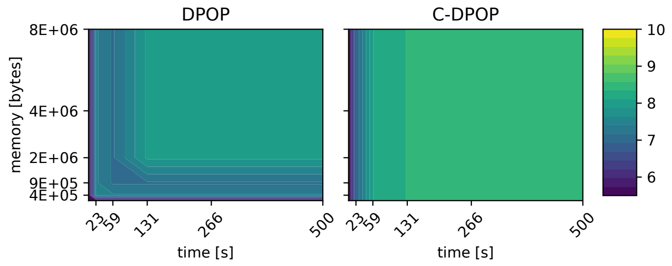


Fig. 4. A comparison of achieved utility for a three agent, ten target problem. Left: the utility of the DPOP algorithm, where the increase in resources (and the number of samples) can be clearly seen to increase the performance. Right: the utility of the C-DPOP algorithm, where the effect of the constant number of samples can be seen in the memory indifference. The performance can be seen to gradually increase when more iterations are possible within the available time.

positioning, which induces the least amount of movement cost and achieves higher sensing utility by being able to sense multiple targets simultaneously.

This property can be clearly seen in Figure 5 where the computation time and memory requirements to achieve a certain resolution are compared. Therefore, given the same amount of resources, C-DPOP is able to achieve a higher resolution.

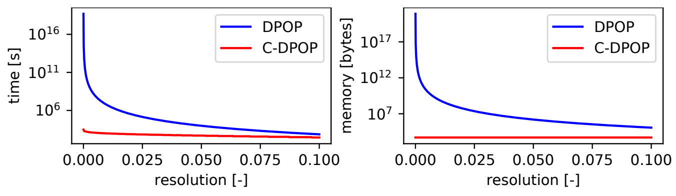


Fig. 5. Required time and memory comparison between DPOP approach and the C-DPOP algorithm for a three agent, ten target problem. It can be seen that the time and memory requirements for DPOP grow exponentially when the required resolution is improved.

VI. CONCLUSION

In this paper, an extension of the Distributed Constraint Optimization Problem called Continuous DCOP (C-DCOP) is proposed to represent variables with continuous domains. The modeling simplicity and versatility of DCOP is thereby extended to include problems with continous variables. Many real-world problems contain inter-agent constraints and limited resources, such as limited computation time and memory. Especially in dynamic environments and close collaboration, these bounds need to be taken into account. For this reason, a solver for the C-DCOP model has been proposed that takes these constraints into account explicitly. The proposed Compression-DPOP (C-DPOP) algorithm is an extension of Distributed Pseudo-tree Optimization Procedure (DPOP) that iteratively samples the continuous domains and dynamically updates the domains after each iteration based on the found optimum.

A mobile sensor coordination problem was used to demonstrate the performance of C-DPOP in comparison with DPOP. Here it was found that the C-DPOP algorithm outperforms DPOP for resource-constrained agents. Higher utility can thus be achieved by using the available resources more effectively.

In future work the sampling method of the continuous domains will be extended from uniform to utility-based, where the utility values of the previous iteration will be used to estimate regions of high utility. The estimation is then refined at every iteration by incorporating the gained information from the samples. Furthermore, we will compare the performance between C-DPOP and other approximate DCOP solvers such as DSA [18]. Additionally, the convergence properties of the C-DPOP algorithm will to be investigated with respect to the compression factor.

REFERENCES

[1] D. M. Sato, A. P. Borges, P. Márton, and E. E. Scalabrin, "I-DCOP: Train Classification Based on an Iterative Process Using Distributed Constraint Optimization," *Procedia Computer Science*, vol. 51, pp. 2297–2306, 2015.
[2] R. Zivan, T. Parash, and Y. Naveh, "Applying max-sum to asymmetric distributed constraint optimization," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 432–439, 2015.
[3] E. A. Sultanik, P. J. Modi, and W. W. C. Regli, "On modeling multiagent task scheduling as a distributed constraint optimization problem," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 247–253, 2007.
[4] M. Pujol-Gonzalez, "Scaling DCOP algorithms for cooperative multiagent coordination," *Constraints*, vol. 20, no. 4, pp. 496–497, 2015.
[5] A. Petcu and B. Faltings, "DPOP: A Scalable Method for Multiagent Constraint Optimization," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 266–271, 2005.
[6] A. Viseras, V. Karolj, and L. Merino, "An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints," in *SAC Symposium on Applied Computing*, pp. 268–275, 2017.
[7] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings, "Decentralised coordination of continuously valued control parameters using the max-sum algorithm," *AAMAS International Conference on Autonomous Agents and Multiagent Systems*, pp. 601–608, 2009.
[8] J. Cerquides, A. Farinelli, P. Meseguer, and S. D. Ramchurn, "A tutorial on optimization for multi-agent systems," *The Computer Journal*, vol. 57, no. 6, pp. 799–824, 2014.
[9] A. R. Leite, F. Enembreck, and J.-P. A. Barthès, "Distributed Constraint Optimization Problems: Review and perspectives," *Expert Systems with Applications*, vol. 41, no. 11, pp. 5139–5157, 2014.
[10] A. Petcu, *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, École Polytechnique Fédérale De Lausanne, 2007.
[11] A. Petcu and B. Faltings, "Approximations in distributed optimization," in *International Conference on Principles and Practice of Constraint Programming*, pp. 802–806, 2005.
[12] I. Brito and P. Meseguer, "Improving DPOP with function filtering," in *AAMAS International Conference on Autonomous Agents and Multi Agent Systems*, pp. 141–148, 2010.
[13] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings, "Bounded approximate decentralised coordination via the max-sum algorithm," *Artificial Intelligence*, vol. 175, no. 2, pp. 730–759, 2011.
[14] E. C. Freuder and M. J. Quinn, "Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1076–1078, 1985.
[15] A. Meisels, *Distributed Search by Constrained Agents: Algorithms, Performance, Communication (Advanced Information and Knowledge Processing)*. 2007.
[16] B. Awerbuch, "A new distributed Depth-First-Search algorithm," *Information Processing Letters*, vol. 20, no. 3, pp. 147–150, 1985.
[17] R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. Sycara, "Distributed constraint optimization for teams of mobile sensing agents," *Autonomous Agents and Multi-Agent Systems*, vol. 29, no. 3, pp. 495–536, 2015.
[18] S. Fitzpatrick and L. Meertens, "Distributed Coordination through Anarchic Optimization," in *Distributed Sensor Networks*, pp. 257–295, Springer, 2003.