Distributed Bayesian: A Continuous Distributed Constraint Optimization Problem Solver

Fransman, J.E.; Sijs, J.; Dol, Henry; Theunissen, Erik; De Schutter, B.H.K.

DOI
10.1613/jair.1.14151

Publication date
2023

Document Version
Final published version

Published in
Journal of Artificial Intelligence Research

Citation (APA)
Fransman, J. E., Sijs, J., Dol, H., Theunissen, E., & De Schutter, B. H. K. (2023). Distributed Bayesian: A Continuous Distributed Constraint Optimization Problem Solver. *Journal of Artificial Intelligence Research*, *76*(1165), 393-433. Article 14151. https://doi.org/10.1613/jair.1.14151

Journal of Artificial Intelligence Research 76 (2023) 393-433 Submitted 08/2022; published 01/2023

Distributed Bayesian: A Continuous Distributed Constraint Optimization Problem Solver

Jeroen Fransman J.E.FRANSMAN@TUDELFT.NL
Delft Center for Systems and Control (DCSC),
Delft University of Technology

Joris Sijs J.SIJS@TUDELFT.NL
Delft Center for Systems and Control (DCSC),
Delft University of Technology

Henry Dol HENRY.DOL@TNO.NL
Netherlands Organisation for Applied Scientific Research (TNO)

Erik Theunissen E.THEUNISSEN@UAV.NL
Netherlands Defence Academy (NLDA)

Bart De Schutter B.DESCHUTTER@TUDELFT.NL
Delft Center for Systems and Control (DCSC),
Delft University of Technology

Abstract

In this paper, the novel Distributed Bayesian (D-Bay) algorithm is presented for solving multi-agent problems within the Continuous Distributed Constraint Optimization Problem (C-DCOP) framework. This framework extends the classical DCOP framework towards utility functions with continuous domains. D-Bay solves a C-DCOP by utilizing Bayesian optimization for the adaptive sampling of variables. We theoretically show that D-Bay converges to the global optimum of the C-DCOP for Lipschitz continuous utility functions. The performance of the algorithm is evaluated empirically based on the sample efficiency. The proposed algorithm is compared to state-of-the-art DCOP and C-DCOP solvers. The algorithm generates better solutions while requiring fewer samples.

1. Introduction

Many real-world problems can be modeled as multi-agent problems in which agents need to assign values to their variables to optimize a global objective characterized by a utility function. Examples include scheduling (Sato, Borges, Márton, & Scalabrin, 2015), mobile sensor coordination (Zivan, Parash, & Naveh, 2015), hierarchical task network mapping (Sultanik, Modi, & Regli, 2007), and cooperative search (Acevedo, Arrue, Maza, & Ollero, 2013). Even though numerous algorithms exist that solve these problems, applying them in practice is often problematic, as complications arise from limitations in communication, computation, and/or memory.

The Distributed Constraint Optimization Problem (DCOP) framework is well suited to model the above-mentioned problems (as detailed in Meisels (2007), Modi, Shen, Tambe, and Yokoo (2005), Petcu and Faltings (2005), Gershman, Meisels, and Zivan (2009), Yeoh and Yokoo (2012)). Within the DCOP framework, a problem is defined based on variables and on utility functions that are aggregated into an objective function. Additionally,

agents assign values to all the variables that are allocated to them. Agents are considered neighbors if their variables are arguments of the same utility function. Neighbors cooperatively optimize their utility functions through the exchange of messages. Within the DCOP framework, variables are constrained by their domains. In other words, a domain defines all possible value assignments of a variable. This explicit definition of the domains of the variables is suitable for problems that are (input) constrained. These domains are considered to be finite and discrete within a DCOP, while problems related to mobile (sensor) platforms are typically characterized by finite continuous domains. Problems with finite continuous domains can be modeled as a Continuous DCOP (C-DCOP), which is equal to a DCOP except for the domain definition. A common approach to solve a C-DCOP with a DCOP solver is to use equidistant discretization of the domains, such as using a grid overlay to define all possible positions of an agent in an area. This process converts the continuous domains into discrete domains. When discretizing a continuous domain, the quality of the solution will depend on the distance between the values, where a smaller distance will allow for a better solution. This results in a trade-off as more values will increase the cardinality of the discrete domains. The increase in cardinality will result in polynomial growth of the search space where the degree of the polynomial is equal to the number of agents. From the overview articles of Leite, Enembreck, and Barthès (2014) and Fioretto, Pontelli, and Yeoh (2018), it is clear that the cardinality of the domains is a major restriction to DCOP solvers. Therefore, solving a C-DCOP by discretization can become intractable for DCOP solvers despite a small number of variables.

The underlying reason for the increase in problem size is that DCOP solvers (implicitly) consider all values within a domain as unrelated to each other. Because of this assumption, it is not possible to efficiently sample the search space. In problems with continuous domains, this assumption does not hold since the utility of values that are close is often similar. By explicitly taking such a relation into account, a C-DCOP solver based on efficient sampling methods can be constructed.

Several C-DCOP solvers have been introduced that, based on initial discretization of the continuous domains, update the discretized values within an iterative optimization method such as local gradient descent. Notable examples are CMS (Stranders, Farinelli, Rogers, & Jennings, 2009), C-CoCoA (Sarker, Arif, Choudhury, & Khan, 2020), PFD (Choudhury, Mahmud, & Khan, 2020), and AC-DPOP (Hoang, Yeoh, Yokoo, & Rabinovich, 2020). In this paper, Bayesian optimization (Mockus, 1989) will be used as it focuses on efficient sampling during optimization, thereby requiring relatively few iterations to closely approach the optimum. This method completely eliminates need for discretization of the domains.

Overall, the contributions of this paper are threefold. Firstly, we introduce an efficient algorithm that uses methods found in Bayesian optimization to solve C-DCOPs called Distributed Bayesian (D-Bay). Secondly, we provide theoretical proof of the convergence of the proposed algorithm to the global optimum of the C-DCOP for utility functions with known Lipschitz constants. Lastly, simulation results are given for randomly generated graphs and sensor coordination problems to compare the sample efficiency of D-Bay to state-of-the-art DCOP and C-DCOP solvers.

The remainder of this paper is organized as follows. Firstly, in Section 2 background information about the DCOP framework is given. In Section 3 relevant literature regarding DCOP solvers is discussed. The Bayesian optimization algorithm is provided in Section 4.

Afterward, we present the novel sampling-based C-DCOP solver called D-Bay in Section 5. The theoretical properties of D-Bay are analyzed in Section 6. Evaluation of D-Bay for sensor coordination problems and random graphs are included in Section 7. Finally, Section 8 summarizes the results and defines future work.

2. Distributed Constraint Optimization Problems

The DCOP framework originates from an extension and generalization of Constraint Satisfaction Problems (CSPs) (Tsang, 1993) towards distributed optimization. A solution for a CSP is defined as the assignment of all variables from (finite) discrete domains such that all hard constraints are satisfied. The CSP framework has been extended from a centralized problem framework to an agent-based distributed problem framework in the work of Yokoo, Durfee, Ishida, and Kuwabara (1998). Within the Distributed-CSP framework, the variables are allocated to agents and the agents coordinate the value assignments among each other.

Additionally, CSP has been generalized into the Constraint Optimization Problem (COP) framework, where the hard constraints are replaced with soft constraints expressed as utility functions. Utility functions return a cost or reward based on the value assignments. Hard constraints are enforced by the utility functions by returning infinite cost (or infinite negative reward). Instead of constraint satisfaction, the goal of a COP is to find assignments that optimize an objective function.

A final extension is the Distributed Constraint Optimization Problem (DCOP) framework, which provides a unified framework that includes a large class of problems by combining the generalization of Distributed-CSP and the extension of COP. A graphical overview of the relations between the problem frameworks can be seen in Figure 1.

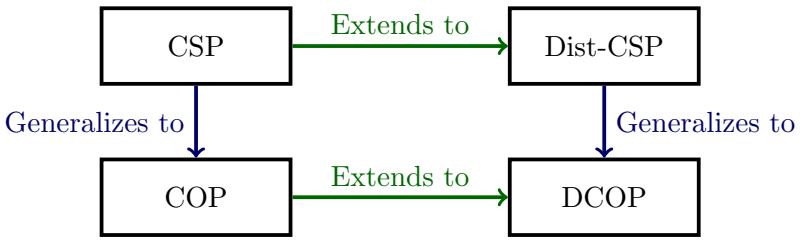


Figure 1: Graphical overview of the relations between the problem frameworks. Adapted from Fioretto et al. (2018).

In the DCOP framework the domains are considered to be discrete, which limits its application to problems with continuous domains. In the current paper, the Continuous DCOP (C-DCOP) framework will be used to overcome this restriction in order to include real-world problems such as cooperative search.

A DCOP is a problem in which an objective function needs to be optimized in a distributed manner through value assignments for all variables. The objective function consists of the aggregate of utility functions, which define a utility value for the value assignments of the variables. All variables within the DCOP are exclusively allocated to agents. An agent is responsible to assign values to all the variables that are allocated to it. Typically,

the number of variables is equal to the number of agents, i.e. every agent is allocated a single variable. The agents cooperate by sending messages to agents with whom they share a utility function. A utility function is shared between agents if their variables are in the arguments of that function. An important aspect of a DCOP is the definition of the domains of the variables. A domain defines all possible values that a variable can be assigned to. In other words, the value assignments are restricted by the domains of the variables. Following the notation of Fioretto et al. (2018), a DCOP is defined by $\mathfrak{D} = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$ where,

- $\mathbf{A} = \{a_1, \ldots, a_M\}$ is the set of agents, where M is the number of agents.
- $\mathbf{X} = \{x_1, \ldots, x_N\}$ is the set of discrete variables, where $N \geq M$ is the number of variables.
- $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$ is the set of domains of all variables, where $\mathbf{D}_i \subseteq \mathbb{R}$ is the finite discrete domain associated with variable x_i. The search space of the DCOP is defined by all possible combinations of all values within the domains as $\mathbf{\Sigma} = \prod_{i=1}^{N} \mathbf{D}_i$, where $\prod$ is the set Cartesian product. The search space of a set of variables ($\mathbf{V} \subseteq \mathbf{X}$) is defined as $\mathbf{\Sigma}_{\mathbf{V}} = \prod_{i \,:\, x_i \in \mathbf{V}} \mathbf{D}_i$.

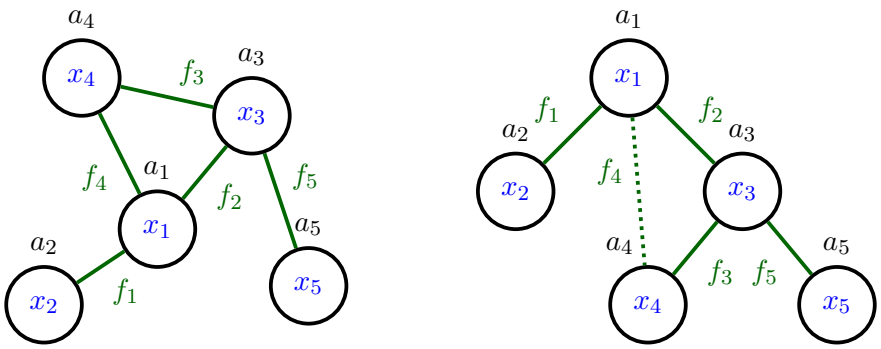
  An assignment denotes the projection of variables onto their domain as $\rho : \mathbf{X} \to \mathbf{\Sigma}$. In other words, for all $x_i \in \mathbf{X}$ if $\rho(x_i)$ is defined, then $\rho(x_i) \in \mathbf{D}_i$. An assignment of a subset of variables is denoted by $\rho_{\mathbf{V}} = \{\rho(x_i) \,:\, x_i \in \mathbf{V}\}$.
- $\mathbf{F} = \{f_1, \ldots, f_K\}$ is the set of utility functions, where K is the number of utility functions. The scope of f_n is denoted as $\mathbf{V}_n \subseteq \mathbf{X}$, where $x_i \in \mathbf{V}_n$ when x_i is an argument of f_n. The optimum of f_n is denoted by $y_n^* = \max_{x \in \mathbf{\Sigma}_{\mathbf{V}_n}} f_n(x)$ with input $x_n^* = \arg\max_{x \in \mathbf{\Sigma}_{\mathbf{V}_n}} f_n(x)$, where $\mathbf{\Sigma}_{\mathbf{V}_n}$ denotes the domain of the utility function.
- $\alpha : \mathbf{X} \to \mathbf{A}$ is a mapping from variables to agents. The agent to which variable x_i is allocated is denoted as $\alpha(x_i)$. A common assumption is that the number of agents is equal to the number of variables, such that $a_i = \alpha(x_i)$ for $i = 1, \ldots, N$. Likewise, the set of agents associated with f_n is denoted by $\alpha(\mathbf{V}_n) = \{\alpha(x) \in \mathbf{A} \mid x \in \mathbf{V}_n\}$.
- η is an operator that combines all utility functions into the objective function. Common options are the *summation* operator ($\sum(\cdot)$) and the *maximum* operator ($\max(\cdot)$). The objective function is defined by $G(\rho) = \underset{f_n \in \mathbf{F}}{\eta} \Big(f_n(\rho_{\mathbf{V}_n}) \Big)$. The optimal value assignment is denoted by $\rho^* \coloneqq \underset{\rho \in \mathbf{\Sigma}}{\arg\max}\, G(\rho)$.

Analogous to the DCOP definition, a Continuous DCOP (C-DCOP) can be defined as a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha, \eta \rangle$. The definitions of $\mathbf{A}$, $\mathbf{F}$, α, and η are identical to their definitions in a DCOP. The differences between a DCOP and a C-DCOP are the definition of the domain set and the variables. All variables in the variable set of a C-DCOP, $\mathbf{X}$, are continuous. The corresponding domain set is defined as $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$, where the domain for variable x_i is defined by a lower bound $\underline{d}_i$ and an upper bound $\overline{d}_i$ as $\mathbf{D}_i = [\underline{d}_i, \overline{d}_i]$.

The relation between the variables and the utility functions is typically represented as a constraint graph. In this representation, the agents are defined as nodes and the edges

implicitly represent the utility functions. A constraint graph is often converted into a pseudo-tree to introduce a hierarchy to the agents. A pseudo-tree is a rooted spanning tree where the subproblems are contained in separate branches. All agents are assigned a single parent, which is an agent higher in the hierarchy. The only exception is the agent on top of the hierarchy, which is denoted as the root of the tree, this agent has no parent. An agent can have multiple children and the agents without children are denoted as the leaves of the tree.

In addition to parent/child relations, the pseudo-tree defines pseudo-parent/pseudo-child relations to indicate relations between agents over multiple hierarchy levels. Typically, the pseudo-tree is used as a communication structure, where agents only communicate between parent and child. In these cases, the pseudo relation allows for (indirect) interaction between pseudo-parent and pseudo-children. A graphical example of the two DCOP representations is given in Figure 2.



(a) Constraint graph (b) Pseudo-tree

Figure 2: Graphical example for two representations of the same DCOP. A node represents an agent a_i and the edges indicate the utility functions f_n based on the variables x_i. In Figure 2a the nodes are unstructured. In Figure 2b the hierarchy is indicated by horizontal layers. The agent at the top (a_1) is the root of the tree, and the bottom agents (a_2, a_4, a_5) are the leaves. The edge (dotted line) between a_1 and a_4 specifies a pseudo-parent/pseudo-child relation.

Producing a pseudo-tree from a constraint graph was introduced in the work of Freuder and Quinn (1985). Various methods exists for the generation of a pseudo-tree (e.g. pseudo-tree ordering (Chechetka & Sycara, 2005), MLSP tree generation (Maheswaran, Tambe, Bowring, Pearce, & Varakantham, 2004), and BFS construction (Chen, He, & He, 2017)). A commonly used method is to produce a depth-first search (DFS) tree from the constraint graph by a DFS traversal. A DFS tree (Petcu & Faltings, 2005) is a special case of a pseudo-tree where the number of edges is equal to the number of edges in the constraint graph. This property ensures that all agents within the DFS tree are also connected in the constraint graph. If the DFS tree is used as a communication structure, this property ensures that agents only communicate if they share a utility function. Various algorithms exist in the literature that creates a DFS tree through a distributed procedure. The interested reader

is referred to the works of Gallager, Humblet, and Spira (1983), Barbosa (1996), Hamadi and Quinqueton (1998), and Awerbuch (1985) for implementation details.

3. Background of DCOP Solvers

In the literature numerous solvers for DCOPs have been proposed; for a detailed overview, the reader is referred to Cerquides, Farinelli, Meseguer, and Ramchurn (2014) and Leite et al. (2014). As noted by Modi et al. (2005), optimally solving a DCOP is NP-hard with regard to the number of variables and the cardinality of their domains. For this reason, complete (optimal) DCOP solvers are often not used in practice. In the literature, a diverse range of incomplete (near-optimal) DCOP solvers exist that trade off solution quality against computational requirements. Such solvers perform well for benchmark problems with domains with low cardinalities, such as graph coloring problems (Modi et al., 2005).

DCOP solvers are unable to directly solve C-DCOPs due to the definition of the domains. However, a C-DCOP can be discretized into a DCOP. One typically discretizes all domains of the C-DCOP using a grid-based approach that converts the continuous domains into discrete domains. This process can arbitrarily increase the cardinality of the domains, thereby rendering the use of DCOP solvers intractable.

For this reason, the development of C-DCOP solvers has recently gained a lot of attention within the literature. Initially, Stranders et al. (2009) introduced the extension of the DCOP framework towards continuous variables (and domains) and proposed a continuous max-sum based (CMS) algorithm to solve C-DCOPs with continuous piecewise linear utility functions. The motivation behind this class of functions was that it can approximate all continuous functions arbitrarily close. In this approach, the domain discretization is replaced with a function approximation that has the same trade-off between resolution and solution quality. In a follow-up paper, Voice, Stranders, Rogers, and Jennings (2010) proposed a hybrid continuous max-sum (HCMS) algorithm without the function approximation. The HCMS algorithm extends the max-sum algorithm by incorporating a continuous non-linear optimization method. Within the algorithm, the domains are discretized and during optimization, the values are updated at every iteration based on a gradient descent method that depends on local utility values. An important parameter of the algorithm is the step size or learning factor with which the values are updated. The authors note that the step size parameter of the algorithm must be adjusted for the given problem, as small values will require numerous iterations while high values could result in overshooting of the optimum.

The concept of discretization of the continuous domains and the iteratively updating of their values is used to extend DCOP solvers of various classes. Based on the taxonomy introduced by Yeoh, Feiner, and Koenig (2010) DCOP solvers can be divided into three classes:

Search-based solvers perform a distributed search over the local search space of the agents. These solvers are based on centralized search techniques such as best-first and depth-first to *reduce the search space* of the problem by exchanging messages between the agents. Examples are ADOPT (Modi et al., 2005), CoCoA (Van Leeuwen, 2017), AFB (Gershman et al., 2009), DSA (Kirkpatrick, Gelatt, & Vecchi, 1983), and DBA (Wittenburg & Zhang, 2003).

Inference-based solvers communicate accumulated information among agents to *reduce the problem size after every message* through dynamic programming methods. Well-known examples of this class of solvers are DPOP (Petcu & Faltings, 2005), the max-sum based algorithm (Rogers, Farinelli, Stranders, & Jennings, 2011), and action GDL (Vinyals, Rodríguez-Aguilar, & Cerquides, 2009).

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures. The probabilistic measures are calculated based on (all) preceding samples to *balance exploration and exploitation* of the global search space. At the time of writing, two sampling-based solvers are found in the literature: DUCT (Ottens, Dimitrakakis, & Faltings, 2017) and Distributed Gibbs (Nguyen, Yeoh, Lau, & Zivan, 2019).

The inference-based DPOP algorithm (Petcu & Faltings, 2005) is extended by Hoang et al. (2020) into several algorithms, where AC-DPOP and the memory-limited variant CAC-DPOP can be applied to C-DCOPs without requirements on the utility functions. The search-based CoCoA algorithm (Van Leeuwen, 2017) is extended in a similar manner by Sarker et al. (2020) into C-CoCoA. Both Stranders et al. (2009) and Sarker et al. (2020) note that the local gradient descent approach cannot guarantee convergence to a global optimum. Initial domain discretization thus remains an important factor in the solution quality. An alternative approach to gradient-based optimization is presented by Choudhury et al. (2020). The proposed Particle Swarm Based F-DCOP (PFD) is based on Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995). PSO is a stochastic optimization technique in which multiple particles are assigned a random position and velocity. The positions represent domain values and the velocities contain implicit derivative information. The PFD algorithm guarantees convergence to a local optimum when the velocity of the best particle is reduced to zero. In conclusion, local gradient-descent-based and PSO-based algorithms will arguably find higher quality solutions compared to their discrete counterparts, but they remain dependent on the initial discretization of the domains and do not guarantee convergence to the global optimum.

In the literature, alternative approaches exist that are capable of extending the inference-based or search-based solvers toward continuous domains. A prominent example is the adaption of dynamic programming for Markov Decision Problems (MDPs) with continuous domains toward C-DCOPs. The interested reader is referred to the work of van Hasselt (2012), and the references therein, for a reinforcement learning approach for learning policies for MDPs with continuous state and action spaces. Additionally, the work of Vianna, Sanner, and de Barros (2014) extends symbolic dynamic programming techniques to solve discrete and continuous state MDPs.

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures, where samples are referred to as (partial) value assignments. Note that in the field of stochastic optimization samples are typically only defined in the context of functions perturbed by noise. The probabilistic measures are used to quantify the probability of finding samples that correspond with function outputs with high utility values. Sampling-based solvers iteratively select samples while taking previous iterations into account to balance exploration and exploitation of the global search space. Sampling-based solvers have not been extended towards the application to C-DCOPs, however, the itera-

tive process allows for the selection of a sample from a continuous domain directly. The elimination of the discretization of the domains combined with the balanced search of the global search space makes sampling-based solvers highly promising to efficiently solve C-DCOPs. Therefore, in Section 4, the sample selection for C-DCOPs is addressed and a novel sampling-based solver for C-DCOPs is introduced in Section 5.

4. Sample Selection for Continuous DCOPs

As mentioned in Section 3, a sampling-based solver will be introduced such that the relation between the value assignment and the corresponding utility will be taken into account within the optimization process. During the optimization process, a sample is defined as a value assignment of a variable. Samples of agents are combined and used as inputs for the utility functions to calculate the corresponding utility values. Within the literature several methods exist for sample generation:

- Upper Confidence Bound (UCB) sampling (Auer, Cesa-Bianchi, & Fischer, 2002), developed for the reduction of regret for K-armed bandit problems (Bather, Berry, & Fristedt, 1986).
- Gibbs sampling (Geman & Geman, 1984), constructed to approximate joint probability distributions in a Markov random field.
- Bayesian sampling (Mockus, 1982), designed for the optimization of utility functions that are computationally expensive to evaluate.

UCB sampling has been applied to DCOPs by Ottens et al. (2017) based on the UCB Applied to Trees (UCT) (Kocsis & Szepesvári, 2006) and HOO (Bubeck, Munos, Stoltz, & Szepesvári, 2011). The resulting algorithm, Distributed UCT (DUCT), generates samples based on previously returned values and the uncertainty (or confidence) over these values. All possible values (of the discrete domains) are sampled at least once before sampling is based on the confidence bounds. Gibbs sampling has been extended by Nguyen et al. (2019) for the optimization of DCOPs based on the mapping of a DCOP to a maximum *a* posteriori (MAP) estimation problem. The MAP is found by approximating a joint probability distribution over all the variables. Samples are generated to approximate the joint probability distribution in a distributed manner. Note that several initial samples are required to accurately represent the desired distribution.

Both UCT and Gibbs sampling share a drawback compared to Bayesian sampling: these methods do not allow for the inclusion of *a* priori knowledge about the utility functions. This allows for Bayesian sampling to generate samples more efficiently than UCT and Gibbs. For this reason, in this paper, Bayesian sampling is extended towards the application to C-DCOPs.

Bayesian sampling is based on Bayesian optimization which is a method to find the global optimum of a function in a sample-efficient manner, i.e. it minimizes the number of required samples. Bayesian optimization consists of two elements: a probabilistic model to approximate a (utility) function $f(\cdot)$, and an acquisition function $q(\cdot)$ to optimally select a new sample x_s, where s denotes the sample index. These two elements are discussed in more detail in Sections 4.1 and 4.2. Every input/output pair, $O_s = (x_s, y_s)$, is included in

the ordered observations set $\mathcal{O}_S = \{O_1, \ldots, O_S\}$, where S is the number of observations and $y_s = f(x_s)$ is the function (utility) value. The observations are used to update the probabilistic model, such that after every new observation the approximation is refined. Based on observations, the probabilistic model is used to estimate a mean function

$$\mu(x, \mathcal{O}) = \mathbb{E}\left[f(x)|\mathcal{O}\right]$$

and the corresponding variance function

$$\sigma^2(x, \mathcal{O}) = \mathbb{E}\left[\left(\left[f(x)|\mathcal{O}\right] - \mu(x, \mathcal{O})\right)^2\right].$$

An overview of the Bayesian optimization algorithm is given in Algorithm 1.

Algorithm 1: Bayesian optimization (Mockus, 1982)

Input : $f(\cdot)$, $q(\cdot)$, S
Output: $\mu(x, \mathcal{O})$, $\sigma^2(x, \mathcal{O})$
`/* Initialize the observation set */`
$\mathcal{O}_0 \coloneqq \emptyset$;
for *s = 1, 2, …, S* **do**
  `/* Select the next sample based on acquisition function */`
  $x_s \coloneqq \arg\max_x q(x|\mathcal{O}_{s-1})$;
  `/* Sample the utility function */`
  $y_s \coloneqq f(x_s)$;
  `/* Augment (and reorder) the observation set */`
  $\mathcal{O}_s \coloneqq \mathcal{O}_{s-1} \cup \{(x_s, y_s)\}$;
  `/* Calculate the mean function and the variance function */`
  $\mu(x, \mathcal{O}) = \mathbb{E}\left[f(x)|\mathcal{O}\right]$;
  $\sigma^2(x, \mathcal{O}) = \mathbb{E}\left[\left(\left[f(x)|\mathcal{O}\right] - \mu(x, \mathcal{O})\right)^2\right]$;
end

4.1 Probabilistic Model

The Gaussian process is a widely used probabilistic model to represent acquired knowledge about a function. More elaborate models exist, but these will often not share the computational benefit of the Gaussian process model. Using the Gaussian process model, a function $f(\cdot)$ is modeled based on a prior mean function $m(x) = \mathbb{E}\left[f(x)\right]$ and a kernel $\kappa(x, x') = \mathbb{E}\left[\left(f(x) - m(x)\right)\left(f(x') - m(x')\right)^{\mathrm{T}}\right]$. The kernel represents the cross-correlation between two values of a variable x, x'. The prior mean function and the kernel contain all (prior) knowledge of $f(\cdot)$. Typically, no prior information about the function is available and the zero function ($m(x) = 0$ for all x) is used as the prior mean function. In such cases, the modeling of the function depends mostly on the choice of the kernel. The Gaussian process model is combined with the observations to construct the joint Gaussian distribution over the function.

From the joint Gaussian distribution, the posterior (distribution) can be found by using the Sherman-Morrison-Woodbury formula (Sherman & Morrison, 1950):

$$P(f(x)|\mathcal{O}) = \mathcal{N}\left(\mu(x|\mathcal{O}), \sigma^2(x|\mathcal{O})\right), \tag{1}$$

where

$$\mu(x|\mathcal{O}) = \boldsymbol{k}(x|\mathcal{O})^{\mathrm{T}} \boldsymbol{K}^{-1}(\mathcal{O}) \boldsymbol{y}(\mathcal{O}), \tag{2}$$

$$\sigma^2(x|\mathcal{O}) = \kappa(x,x) - \boldsymbol{k}(x|\mathcal{O})^{\mathrm{T}} \boldsymbol{K}^{-1}(\mathcal{O}) \boldsymbol{k}(x|\mathcal{O}), \tag{3}$$

and $\mathcal{N}$ denotes the normal distribution, $\boldsymbol{K}(\mathcal{O})$ is the Gramian matrix of the kernel, defined by $(\boldsymbol{K})_{i,j} = \kappa(x_i, x_j)$ for all $i, j \in \{1, \ldots, S\}$, $\boldsymbol{k}(x|\mathcal{O}) = [\kappa(x_1, x), \ldots, \kappa(x_S, x)]^{\mathrm{T}}$ denotes the *cross-correlation* vector between the observations and x, and $\boldsymbol{y}(\mathcal{O}) = [y_1, \ldots, y_S]^{\mathrm{T}}$ denotes the observation value vector. The (posterior) mean and variance functions of the probabilistic model are denoted as $\mu(\cdot)$ and $\sigma^2(\cdot)$, respectively. Note that the posterior distribution contains the estimate of the function based on both the prior knowledge and the observations.

A wide range of kernels for Gaussian processes exist in the literature and the interested reader is referred to the work of Duvenaud, Nickisch, and Rasmussen (2011) for an overview of constructing kernels. An important kernel property is the ability to estimate every continuous function up to a required resolution given a sufficient number of observations. A kernel that possesses this property is called a universal kernel. In the work of Micchelli, Xu, and Zhang (2006), the conditions for a kernel to be universal in terms of properties of its features are given. The most commonly used universal kernel is the squared exponential kernel. A general description of the kernel and its properties is given by Vert, Tsuda, and Schölkopf (2004). A drawback of the squared exponential kernel is that it can result in over-smooth approximations. For this reason, the Matérn kernel (Minasny & McBratney, 2005) is often used, since it can trade off differentiability and smoothness. In practice, the choice for a kernel depends on the properties of the function that needs to be approximated. All kernels have parameters that can be used to adjust their properties, such as smoothness and scaling. If information about $f(\cdot)$ is available, this should be incorporated in the selection of the kernel and its parameters. Typically, it is assumed that no information about $f(\cdot)$ is available, and then, as noted by Rasmussen and Williams (2006), the selection of the parameters is non-trivial. For this reason, the selection of parameters is often treated as a separate optimization problem (MacKay, 1992). It is commonly solved by using the maximum likelihood problem for which automatic relevance detection (MacKay, 1994) is a widely used algorithm.

4.2 Acquisition Function

The selection of the next sample is the result of the optimization of an acquisition function $q(\cdot)$, defined by

$$x_s = \arg\max_x q(x|\mathcal{O}).$$

The acquisition function depends on the posterior distribution in Equation (1) and thereby on all previous observations. Two commonly used acquisition functions are the probability

of improvement function (Kushner, 1964) and the expected improvement function (Mockus, Tiesis, & Zilinskas, 1978). The probability of improvement function considers the *probability* of finding a sample of which its value is larger than the maximum observed value. The maximum observed value is defined as

$$y^+ = \max\{y_s : (x_s, y_s) \in \mathcal{O}\}.$$

The corresponding maximum sample is defined as $x^+ = \{x_s : (x_s, y_s) \in \mathcal{O} | y_s = y^+\}$. As noted by Brochu, Cora, and de Freitas (2010), the probability of improvement function focuses solely on the exploitation of already observed samples. To balance the exploration of the search space and exploitation of the observations, the expected improvement function will be used in this chapter. The expected improvement function chooses the sample based on the *expected value* of the next observation. The interested reader is referred to Brochu et al. (2010) for a comparison of the two acquisition functions and more details. The expected improvement function can be written in closed form in terms of the mean and the deviation function of the probabilistic model as

$$q_{\mathrm{EI}}(x, \xi|\mathcal{O}) = \begin{cases} z(x, \xi|\mathcal{O})\Phi\left(\frac{z(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})}\right) + \sigma(x|\mathcal{O})\phi\left(\frac{z(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})}\right) & \text{if } \sigma(x|\mathcal{O}) > 0 \\ 0 & \text{if } \sigma(x|\mathcal{O}) = 0 \end{cases} \tag{4}$$

$$z(x, \xi|\mathcal{O}) = \mu(x|\mathcal{O}) - \left(y^+ + \xi\right) \tag{5}$$

where $\Phi(\cdot)$ is the Gaussian cumulative distribution function, $\phi(\cdot)$ is the Gaussian probability density function, and ξ is a design parameter. The design parameter can be used to trade off exploration and exploitation. As noted by Lizotte, Greiner, and Schuurmans (2012), even a value as low as $\xi = 0$ will not result in a solely exploiting sampling behavior.

5. The Distributed Bayesian Algorithm

In the previous sections, background information has been given about the DCOP framework and Bayesian optimization. In this section, the novel sampling-based C-DCOP solver Distributed Bayesian (D-Bay) is presented. This solver is capable of directly solving C-DCOPs without discretization of the domains. D-Bay uses Bayesian optimization as the probabilistic measure to optimize the sample selection. The overall procedure is similar to state-of-the-art sampling-based solvers, e.g. DUCT (Ottens et al., 2017) and Sequential Distributed Gibbs (SD-Gibbs) (Nguyen et al., 2019).

Sampling-based solvers coordinate the sampling of the global search space guided by probabilistic measures to balance exploration and exploitation. The general outline of sampling-based solvers is as follows. Based on a pseudo-tree representation of the C-DCOP, the variables and utility functions are allocated to the agents. Afterward, two consecutive phases are iteratively repeated until a termination condition is satisfied. The first phase, the sampling phase, is top-down and starts from the root agent. The root starts the sampling phase by selecting a sample for all its variables. A sample can be viewed as a temporary value assignment of a variable. The sample is sent as a **sample** message to all the children of the agent. Upon receiving this message, an agent samples its variables and adds these samples to the **sample** message before sending it to its children. This process continues until the leaf agents are reached.

When the leaf agents are reached, the utility phase is initiated. This second phase is bottom-up and starts from the leaf agents. Based on the allocated utility functions, the agents calculate the utility (value) based on the **sample** message and the assignments of their variables. This utility is encoded within a **utility** message and sent to the parent of the agent. Upon receiving a **utility** message, an agent calculates the utility of its allocated utility functions. The resulting utility value is aggregated with the utility value of the received message before sending a **utility** message to its parent. This phase finishes when the root agent received a **utility** message from all its children. This moment marks the end of a single iteration. At this time, all agents have obtained the utility value associated with the sample of their variables. This information is used by the agents to update their probabilistic models and thereby the selection of their sample at the next iteration.

The main difference between sample-based solvers is in the method of selecting additional samples. The probabilistic measure in DUCT is based on confidence bounds of the utility of the samples and selects samples to improve these bounds. The agents store the utility for all previous samples during optimization. The Distributed Gibbs algorithm selects samples based on joint probability distributions and only keeps track of the differences between the utility values of the samples as a termination criterion. This makes Distributed Gibbs more memory efficient compared to DUCT.

Both algorithms are DCOP solvers and have a runtime complexity that is linear in the cardinality of the largest domain (Fioretto et al., 2018, Table 4). Therefore, both Distributed Gibbs and DUCT suffer from the discretization of continuous domains and are not suitable for continuous DCOPs.

An additional disadvantage of both solvers is the non-determinism with regard to the utility value of a sample. This is caused by the consecutive sampling and utility phases since within an iteration all agents sample a single value from their local search space. In other words, the same **sample** message can result in different **utility** messages when the children of an agent select different samples for their variables.

To remove the non-determinism, the sampling and utility phase in D-Bay will be restricted to parents and children instead of the entire pseudo-tree. To be more precise, when a child receives a **sample** message it will first iterate between its children before sending a **utility** message to its parent. This will guarantee that the **utility** message in response to a **sample** message will always be identical.

D-Bay as described in Algorithm 2 (Appendix A) involves four phases:

(1) Pseudo-tree construction: The agents create a pseudo-tree from the constraint graph of the C-DCOP. Afterwards, every agent a_i knows its parent/children sets ($\mathbf{P}_i/\mathbf{C}_i$) and pseudo-parents/pseudo-children sets ($\mathbf{PP}_i/\mathbf{PC}_i$), where $\mathbf{P}_i, \mathbf{PP}_i, \mathbf{C}_i, \mathbf{PC}_i \subset \mathbf{A}$. The pseudo-tree is used as the communication structure in which agents only communicate with agents with whom they share a parent/child relation. Note that the agents only have local information on the pseudo-tree.

(2) Allocation of utility functions: Similar to the allocation of variables, all utility functions in $\mathbf{F}$ are exclusively allocated to the agents. Every agent a_i constructs two separate function sets based on the variables of the agent and the variables of its (pseudo-)parents. Firstly, the utility function set $\mathbf{F}_{a_i} = \{f_n \in \mathbf{F} : \alpha(\mathbf{V}_n) = \{a_i\}\}$, which only depends on the agent itself. Secondly, the shared utility function set, $\mathbf{F}_{\mathbf{P}_i} = \{f_n \in \mathbf{F} : (a_i \in \alpha(\mathbf{V}_n)) \wedge (\alpha(\mathbf{V}_n) \cap (\mathbf{P}_i \cup \mathbf{PP}_i) \neq \emptyset)\}$, involves the agent and its (pseudo-)parents. These two function sets are combined as $\mathbf{F}_i = \mathbf{F}_{a_i} \cup \mathbf{F}_{\mathbf{P}_i}$.

(3) Sample propagation: In this phase, every agent optimizes its variables through the Bayesian optimization method and exchanges **sample** and **utility** messages. By doing so, the assignments of the variables will converge to the global optimum of the objective function as will be shown in Section 6.2. The variables of a_i are defined as $\mathbf{X}_i = \{x_j \in \mathbf{X} \mid a_i \in \alpha(x_j)\}$. The variables are optimized based on the aggregate utility of all functions in set $\mathbf{F}_i$ and the functions of its children ($f_n \in \mathbf{F}_k$ for all $a_k \in \mathbf{C}_i$). Consequently, the aggregate utility values obtained by the root agent hold the utility values of the objective function.

Since a sample from (pseudo-)parents is required to calculate the utility of the functions in $\mathbf{F}_{\mathbf{P}_i}$, every agent a_i waits for a **sample** message from its parent. The phase is therefore initiated by a **sample** message from the root agent. The sample propagation phase finishes when a convergence threshold is reached by the root agent. Upon receiving **sample** message $\mathfrak{S}_j$ from its parent a_j, agent a_i samples its variables with respect to the functions in set $\mathbf{F}_i$. The samples are selected through the optimization of an acquisition function. Note that the acquisition function depends on both a kernel and all preceding samples. If the agent is a leaf agent, the agent can optimize its variables without considering the impact of its assignments on other agents. The agents with children augment the **sample** message of their parent with their sample as $\mathfrak{S}_i = \mathfrak{S}_j \cup \{\rho_{\mathbf{X}_i}\}$ and send this message to their children.

The agent then waits until it has received all **utility** messages from its children. Only then is the agent able to compute the aggregate utility and return a **utility** message to its parent. Note that the aggregate utility represents the optimal utility for the sample of the agent and all its (pseudo-)children. A **utility** message is defined as $\mathfrak{U}_i^j = \eta\left(\mathfrak{U}_i, \hat{\mathfrak{U}}_i\right)$, where $\mathfrak{U}_i = \min_{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}} \eta_{f_n \in \mathbf{F}_i}\left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j)\right)$ and $\hat{\mathfrak{U}}_i = \eta_{a_k \in \mathbf{C}_i}\left(\mathfrak{U}_k^i\right)$ define the utility and the aggregated child utility, respectively. A graphical overview of the sample propagation phase is shown in Figure 3. Additionally, a partial trace of the Bayesian optimization is shown in Figure 4.

(4) Assignment propagation: The final phase is the assignment propagation phase, in which the root agent a_1 sends the final assignment of all its variables to its children as a **final** message $\hat{\mathfrak{S}}_1 = \{\hat{\rho}_{\mathbf{X}_1}\}$. Based on these assignments, the children assign their variables to the value corresponding to the optimal utility value. Afterwards, every agent adds its assignments to the **final** message as $\hat{\mathfrak{S}}_i = \{\hat{\rho}_{\mathbf{X}_i}\} \cup \hat{\mathfrak{S}}_j$. After the leaf agents have received a **final** message, all agents have completed their local assignments $\hat{\rho}_{\mathbf{X}_i}$. Note that typically no agent has information of the complete assignment, $\hat{\rho} = \{\hat{\rho}_{\mathbf{X}_i} \ : \ i = 1, \ldots, M\}$.

Proposition 1. The memory complexity of each agent in D-Bay is $O(S)$, where S is the number of samples at every iteration.

Proof. The optimization of the variables of an agent is restarted every time a **sample** message from the parent is received. An agent only needs to store the utility of the values based on the current (local) iteration to send the *best* utility value back to its parent, thereby restricting the memory requirement per agent to $O(S)$. □

Proposition 2. The maximal message size for all messages in D-Bay is $O(t)$, where t is the maximal depth of the tree.

Proof. The **utility** message has a fixed size of $O(1)$ as it contains a single utility value related to the current sample message $\mathfrak{S}$. Both the **sample** message and the **final** message are appended with the sample of the agent before it is sent to the children of the agent. This limits the size of these messages to $O(t)$. □

Proposition 3. The total number of messages sent by an agent in D-Bay is $O(cS^t)$, where t is the depth of the tree, S is the number of samples and c denotes the largest number of children.

Proof. When agent a_i receives a **sample** message from its parent it generates S samples that are sent to all its children $\mathbf{C}_i$. This process continues until the leaves of the pseudo-tree are reached and will bind the number of messages for an agent to $O(cS^t)$. □

Proposition 4. The maximal runtime complexity of each agent in D-Bay is $O(S^t)$, where t is the depth of the tree.

Proof. For every **sample** message an agent receives it optimizes the value of its next sample through Bayesian optimization a total of S times. The runtime complexity of the agents is greatest for the leaf agents and therefore the maximum number of samples for an agent is $O(S^t)$. □

In the next section, the convergence of D-Bay to the global optimum of a C-DCOP is analyzed. D-Bay utilizes Bayesian optimization for the sample selection within the sample propagation phase. For that reason, the performance of D-Bay depends highly on the properties of the Bayesian optimization method. As mentioned in Section 4, Bayesian optimization consists of the combination of a kernel and an acquisition function. Therefore, the analysis is focused on the selection of the kernel, the acquisition function, and their parameters.

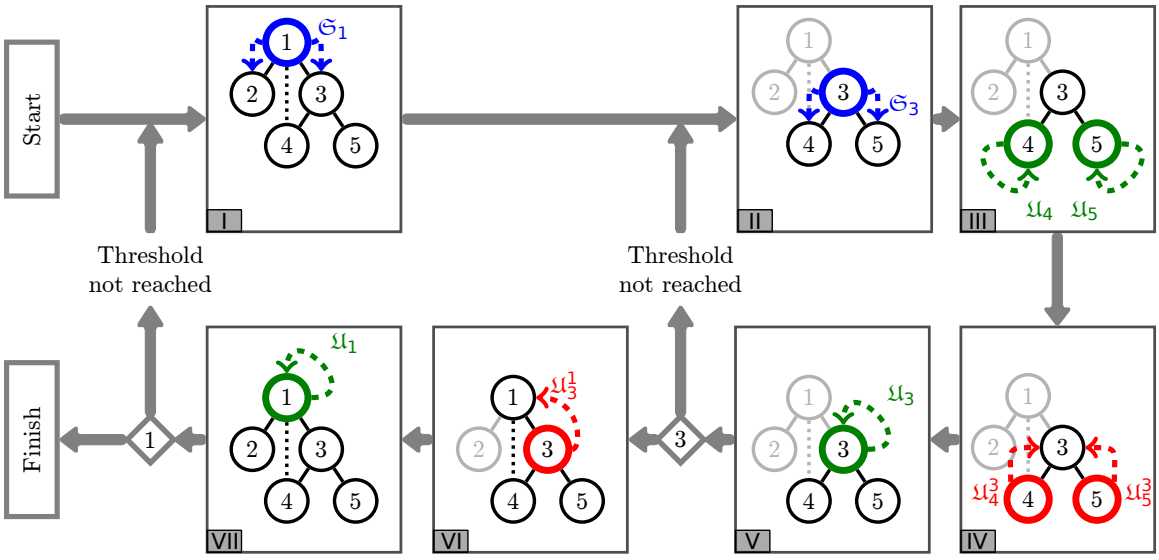
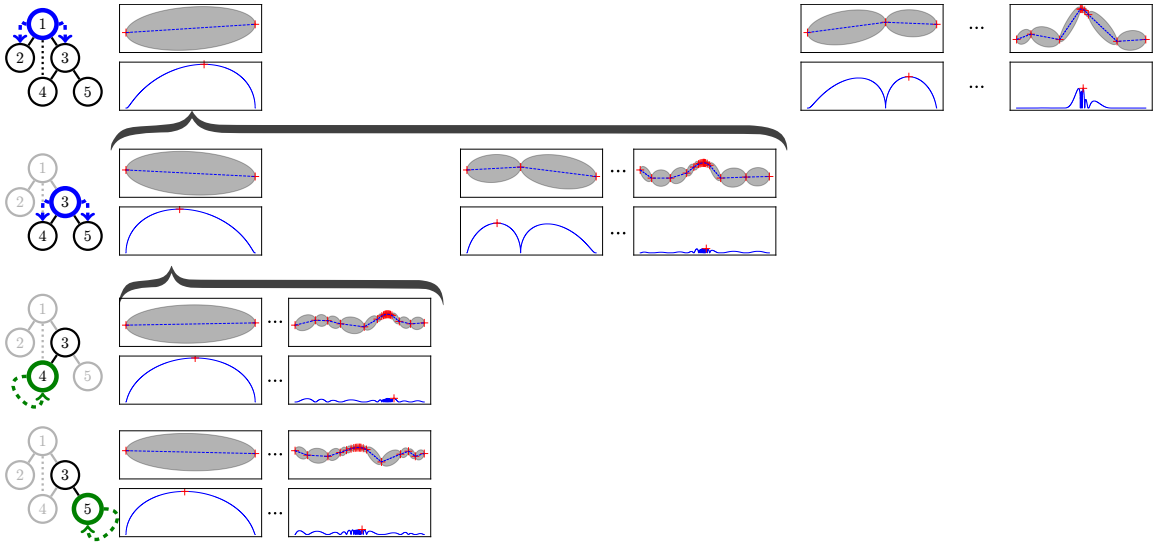
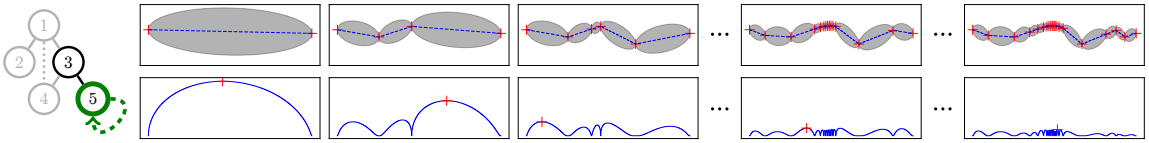


Figure 3: Graphical overview of the sample propagation phase of D-Bay. Agents are indicated by circles labeled with an agent index, and utility functions are shown as black lines. Starting from the root a_1 (I), **sample** message $\mathfrak{S}_1$ is sent to its children (a_2, a_3). Subsequently, agent a_3 will send **sample** message $\mathfrak{S}_3$ to its children (II). After iterating between its children and calculating its local utility (III), agent a_3 combines all utilities (IV) and checks its threshold (V). The check for the threshold is indicated by an annotated grey diamond. If the threshold is reached the agent a_3 sends **utility** message $\mathfrak{U}_3^1$ to its parent (VI). This process is repeated when the root a_1 sends another **sample** message and finishes when a_1 the convergence threshold or the number of samples is reached (VII). Note that the interactions between a_1 and a_2 are not illustrated for brevity.



(a) Partial trace for simple DCOP during sample propagation phase. During the optimization, the children sample their variables based on the **sample** message of a parent. This dependency is indicated by the curly brackets, where the samples associated with a parent sample are contained within the curly bracket.



(b) Detailed view for agent a_5.

Figure 4: Graphical examples of a partial trace of the Bayesian optimization process. The agent associated with the trace is shown on the left. The trace consists of the utility function approximation (top) and the acquisition function (bottom). The function approximation shows the samples (+), the mean (---), and the standard deviation (■). The acquisition function shows both the values (—) and its optimum (+). The optimum of the acquisition function determines the next sample of the objective function.

6. Theoretical Analysis of D-Bay

This section analyses the convergence of D-Bay to the global optimum of a C-DCOP in two parts. Firstly, in Section 6.1, the convergence to the global optimum of the utility functions within the sampling propagation phase is proven. It shows that if the Lipschitz constant of the utility functions is known, the convergence to the global optimum can be guaranteed through the appropriate selection of the kernel and the acquisition function. In this paper, all utility functions are assumed to be Lipschitz continuous with known Lipschitz constant. A utility function $f(\cdot)$ is Lipschitz continuous with Lipschitz constant L_f if

$$|f(x_i) - f(x_j)| \leq L_f|x_i - x_j| \qquad \forall x_i, x_j \in \text{dom}(f) \tag{6}$$

where $\text{dom}(f)$ denotes the domain of the utility function.

Secondly, in Section 6.2, the convergence of D-Bay to the global optimum of the objective function based on the global optima of the utility functions is proven. This analysis focuses on the assignment propagation phase. The two parts of the analysis are combined to prove the convergence of D-Bay to the global optimum of C-DCOPs with utility functions with known Lipschitz constants.

6.1 Convergence of Bayesian Optimization Based on Lipschitz Continuous Functions

As shown by Törn and Žilinskas (1989), the convergence to the global optimum of a function by Bayesian optimization can only be guaranteed through *dense* sampling of the domain of the function. For this reason, within the Bayesian optimization method, the acquisition function will need to produce dense samples. In the work of Vazquez and Bect (2010, Theorem 6), the expected improvement acquisition function, given by Equation (4), is proven to produce dense observations within its search region. The search region is defined in Definition 1.1.

Definition 1.1. The search region of the acquisition function $q_{\text{EI}}(\cdot)$ (based on $f(\cdot)$ and $\mathcal{O}$) is defined by

$$\mathcal{S} = \{x \in \text{dom}(f) : q_{\text{EI}}(x, \xi|\mathcal{O}) > 0\}.$$

As a consequence of the dense sample generation property, $\mathcal{S}$ will converge to an empty set when the number of samples goes to infinity. Therefore, since the next sample is chosen from the search region ($x_s \in \mathcal{S}$), the global optimum (x^*) will be sampled for $s \to \infty$ if $x^* \in \mathcal{S}$. In other words, if the optimum inclusion ($x^* \in \mathcal{S}$) property holds, then global convergence is guaranteed.

Definition 1.2. The upper bound function $\bar{f}(x|\mathcal{O})$ of (a Lipschitz continuous) function $f(\cdot)$ over all observations in $\mathcal{O}$ is defined by

$$\bar{f}(x|\mathcal{O}) = \min\{L_f|x - x_s| + y_s : (x_s, y_s) \in \mathcal{O}\} \qquad \forall x \in \text{dom}(f).$$

Definition 1.3. The upper bound region of $f(\cdot)$ holds all values of x for which the upper bound function $\bar{f}(\cdot)$ (Definition 1.2) is larger than the maximum observed value y^+ and is defined by

$$\mathcal{U} = \{x \in \text{dom}(f) : \bar{f}(x|\mathcal{O}) > y^+\}.$$

To find the conditions for which the optimum inclusion holds, the upper bound region set is introduced. The upper bound region set $\mathcal{U}$ (Definition 1.3) is based on the upper bound function $\bar{f}(\cdot)$ (Definition 1.2). Note that by definition, $\bar{f}(x) \geq f(x)$ for all x and $\mathcal{U}$ does not include any observations in $\mathcal{O}$ since $\bar{f}(x|\mathcal{O}) = y_s \leq y^+$ for all x_s. As shown in Lemma 1.1, this region is guaranteed to include the global optimum if the optimum has not already been observed. A graphical example of sets $\mathcal{U}$ and $\mathcal{S}$, and $\bar{f}(\cdot)$ can be seen in Figure 5.

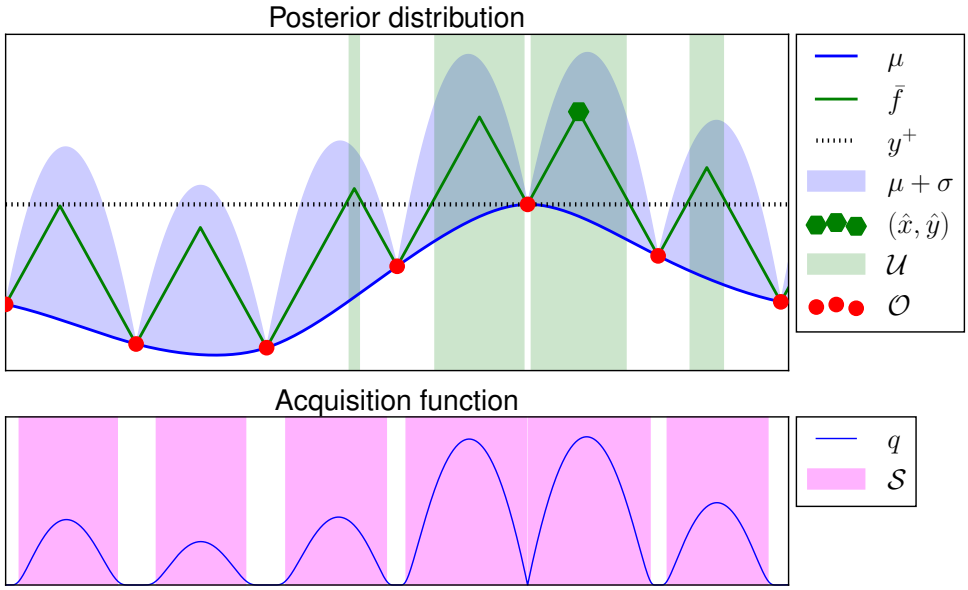


Figure 5: Graphical overview of the sets $\mathcal{U}$, $\mathcal{S}$, and the upper bound function $\bar{f}(\cdot)$ based on the observations $\mathcal{O}$.

Lemma 1.1. *The upper bound region $\mathcal{U}$ includes the optimum sample x^* if it has not been observed. Formally, if $x^+ \neq x^*$, then $x^* \in \mathcal{U}$.*

Proof. By Definition 1.2 and Equation (6), the value of the upper bound function at the optimal sample is larger or equal to the optimal value. Formally, $\bar{f}(x^*|\mathcal{O}) \geq y^*$. If the optimal sample has not been observed ($x^* \neq x^+$), then $y^* > y^+$. Consequently, $\bar{f}(x^*|\mathcal{O}) > y^+$. Therefore, by definition of the upper bound region (Definition 1.3) the optimal sample is included ($x^* \in \mathcal{U}$). □

Based on the definition of the upper bound region, the optimum inclusion is satisfied when the set inclusion $\mathcal{U} \subseteq \mathcal{S}$ holds. The conditions for the set inclusion are given in two parts. Firstly, in Lemma 1.2 it is shown that if for all samples where the sum of the mean and deviation function is greater or equal to the highest sampled value, the sample is included in the search region set. Secondly, by Definition 1.3, the upper bound function defines all samples that are within the upper bound region set. By combining both conditions, we find that if the sum of the mean and deviation function is greater than the upper bound function, then $\mathcal{U} \subseteq \mathcal{S}$, as shown in Lemma 1.3.

Lemma 1.2. *If $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq y^+ + \xi$ and $\sigma(x|\mathcal{O}) > 0$ then $x \in \mathcal{S}$.*

Proof. By Definition 1.1, $x \in \mathcal{S}$ if $q_{\text{EI}}(x, \xi|\mathcal{O}) > 0$. Let $\sigma(x|\mathcal{O}) > 0$ and define $w(x, \xi|\mathcal{O}) = \frac{z(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})}$ and through substitution rewrite Equation (4) as

$$q_{\text{EI}}(x, \xi|\mathcal{O}) = z(x, \xi|\mathcal{O})\Phi\Big(w(x, \xi|\mathcal{O})\Big) + \sigma(x|\mathcal{O})\phi\Big(w(x, \xi|\mathcal{O})\Big). \tag{7}$$

Since $\sigma(x|\mathcal{O}) > 0$, we find $q_{\text{EI}}(x, \xi|\mathcal{O}) > 0$ if $\frac{q_{\text{EI}}(x,\xi|\mathcal{O})}{\sigma(x|\mathcal{O})} > 0$ where

$$\frac{q_{\text{EI}}(x, \xi|\mathcal{O})}{\sigma(x|\mathcal{O})} = w(x, \xi|\mathcal{O})\Phi\Big(w(x, \xi|\mathcal{O})\Big) + \phi\Big(w(x, \xi|\mathcal{O})\Big).$$

Define $h(w) = w\Phi(w) + \phi(w)$. Then since $\Phi'(w) = \phi(w)$ and $\phi(w) = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}w^2}$, we find $h'(w) = \Phi(w) + w\phi(w) - w\phi(w) = \Phi(w)$. Let $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq y^+ + \xi$, then $z(x, \xi|\mathcal{O}) \geq -\sigma(x|\mathcal{O})$ and $w(x, \xi|\mathcal{O}) \geq -1$. For w in the interval $(-1, \infty]$ we find

$$h(w) = \int_{-1}^{w} h'(v)\mathrm{d}v + h(-1) = \int_{-1}^{w} \Phi(v)\mathrm{d}v - \Phi(-1) + \phi(-1) > 0,$$

since $\Phi(w) > 0$ for finite inputs, and $-\Phi(-1) + \phi(-1) > 0$.
Therefore, if $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq y^+ + \xi$ and $\sigma(x|\mathcal{O}) > 0$, then $q_{\text{EI}}(x, \xi|\mathcal{O}) > 0$ and $x \in \mathcal{S}$. □

Lemma 1.3. *If $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) + \xi$ for all $x \in \text{dom}(f)$, then $\mathcal{U} \subseteq \mathcal{S}$.*

Proof. As shown in Lemma 1.2, if $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq y^+ + \xi$ and $\sigma(x|\mathcal{O}) > 0$, then $x \in \mathcal{S}$. By definition of $\mathcal{U}$, for all $x \in \mathcal{U}$ we find $\bar{f}(x|\mathcal{O}) > y^+$. Additionally, for all $x \in \mathcal{U}$ we find $\sigma(x|\mathcal{O}) > 0$, since $\sigma^2(x|\mathcal{O}) = 0$ only if $x = x_s$ and by definition $x_s \notin \mathcal{U}$. Therefore, if $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) + \xi$, then $x \in \mathcal{S}$ for all $x \in \mathcal{U}$. □

Note that $\mu(\cdot)$ and $\sigma(\cdot)$ depend on the kernel and $\bar{f}(\cdot)$ depends on the Lipschitz constant. This raises the question of which (type of) kernel is capable of explicitly associating its mean function and variance function to the Lipschitz constant.

An answer can be found in the work of Ding and Zhang (2018), where Markovian class kernels are introduced as kernels that possess a Markovian posterior distribution. The value of a Markovian posterior distribution for a certain input value only depends on the observations surrounding that value. This property is beneficial as the upper bound function, which is directly related to the Lipschitz constant, possesses the same property. An additional benefit of this class of kernels is that the elements of $\boldsymbol{K}^{-1}(\mathcal{O})$ can be expressed analytically. This removes the need of inversion of a matrix of which the size grows with the number of observations, since $\boldsymbol{K}(\mathcal{O}) \in \mathbb{R}^{S\times S}$. As noted by Rasmussen and Williams (2006), this inversion is considered a major restriction to the practical application of Bayesian optimization. In general, a Markovian class kernel is defined by

$$\kappa(x_i, x_j) = \lambda^2\Big(p(x_i)g(x_j)\mathbb{I}_{x_i \leq x_j} + p(x_j)g(x_i)\mathbb{I}_{x_i > x_j}\Big)$$

for some function $p(\cdot)$ and $g(\cdot)$, where $\mathbb{I}(\cdot)$ is the indicator function and λ is the kernel scale parameter. The observations are (re)ordered after every new observation, such that for

scalar arguments $x_1 \leq x_2 \leq \cdots \leq x_S$. The mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot)$ of the posterior on the interval between observations for a kernel of this class is defined by,

$$\mu_s(x|\mathcal{O}) = \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathcal{O})\boldsymbol{K}_s^{-1}(\mathcal{O})\boldsymbol{y}_s(\mathcal{O}), \tag{8}$$

$$\sigma_s^2(x|\mathcal{O}) = \kappa(x,x) - \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathcal{O})\boldsymbol{K}_s^{-1}(\mathcal{O})\boldsymbol{\kappa}_s(x,\mathcal{O}), \tag{9}$$

for $x \in [x_{s-1}, x_s]$, where

$$\boldsymbol{\kappa}_s(x,\mathcal{O}) = \begin{bmatrix} \kappa(x_1,x) & \ldots & \kappa(x_{s-1},x) & \kappa(x_s,x) & \kappa(x_{s+1},x) & \ldots & \kappa(x_S,x) \end{bmatrix}^{\mathrm{T}},$$

$$\boldsymbol{y}_s(\mathcal{O}) = \begin{bmatrix} y_1 & \ldots & y_{s-1} & y_s & y_{s+1} & \ldots & y_S \end{bmatrix}^{\mathrm{T}},$$

and $\boldsymbol{K}_s^{-1}(\mathcal{O})$ is a tridiagonal matrix of appropriate dimensions where the tridiagonal elements of $\boldsymbol{K}_s^{-1}(\mathcal{O})$, for $S \geq 3$ and if $\boldsymbol{K}_s(\mathcal{O})$ is nonsingular, are given by

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s} = \begin{cases} \frac{\lambda^{-2}p(x_2)}{p(x_1)\left(p(x_2)g(x_1)-p(x_1)g(x_2)\right)}, & \text{if } s = 1, \\ \frac{\lambda^{-2}\left(p(x_{s+1})g(x_{s-1})-p(x_{s-1})g(x_{s+1})\right)}{\left(p(x_s)g(x_{s-1})-p(x_{s-1})g(x_s)\right)\left(p(x_{s+1})g(x_s)-p(x_s)g(x_{s+1})\right)}, & \text{if } s \in \{2,\ldots,S-1\}, \\ \frac{\lambda^{-2}g(x_{S-1})}{g(x_S)\left(p(x_S)g(x_{S-1})-p(x_{S-1})g(x_S)\right)}, & \text{if } s = S, \end{cases}$$

and

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s-1,s} = (\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s-1} = \frac{-\lambda^{-2}}{\left(p(x_s)g(x_{s-1}) - p(x_{s-1})g(x_s)\right)}, \ s = 2,\ldots,S.$$

All other elements of $\boldsymbol{K}_s^{-1}(\mathcal{O})$ are equal to zero.

Next, we show that for the Dirichlet kernel, as introduced by Ding and Zhang (2018), the inequality of Lemma 1.3 will hold for all observations if the kernel scale λ is chosen appropriately. This kernel is selected over other Markovian class kernels because of its simplicity. The Dirichlet kernel defined by

$$\kappa_{\mathrm{d}}(x_i,x_j) = \lambda^2 \min(x_i,x_j)(1-\max(x_i,x_j)) \tag{10}$$

for $x_i, x_j \in [0,1]$. Note that for κ_{d}, we find that $p(x) = x$ and $g(x) = (1-x)$. The mean function, given by Equation (8), and the variance function, given by Equation (9), corresponding to the Dirichlet kernel in the interval $[x_{s-1}, x_s]$ can be written as

$$\mu_s(x|\mathcal{O}) = \frac{y_{s-1}(x_s - x) + y_s(x - x_{s-1})}{x_s - x_{s-1}}, \tag{11}$$

$$\sigma_s^2(x|\mathcal{O}) = \lambda^2 \frac{-(x_s - x)(x_{s-1} - x)}{x_s - x_{s-1}}. \tag{12}$$

The derivation of Equations (11) and (12) can be found in Appendix B. It is important to note that both the mean function and the variance function on the interval $[x_{s-1}, x_s]$ only depend on the observations at the boundaries of the interval.

Based on Equations (11) and (12), the inequality of Lemma 1.3 holds if

$$\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) + \xi \text{ for } x \in [x_{s-1}, x_s] \tag{13}$$

for all $s \in \{1, \ldots, S\}$, given $x_1 = 0$ and $x_S = 1$. In other words, by using the Dirichlet kernel, instead of analyzing the inequality in Lemma 1.3 over the entire domain of the function, it is sufficient to analyze Equation (13) on the intervals between the observations.

Now that the acquisition function and the kernel have been selected, we need to find their parameters (ξ and λ) such that the inequality of Equation (13) holds. These parameters can be appropriately chosen based on the Lipschitz constant as given in Theorem 1.4.

Theorem 1.4. *For a function $f(\cdot)$ with known Lipschitz constant L_f and* $\mathrm{dom}(f) = [0, 1]$*, the Dirichlet kernel κ_{d}, and $S \geq 3$, where $x_1 = 0$ and $x_S = 1$, will yield $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for all $x \in \mathrm{dom}(f)$ if $\lambda \geq L_f$.*

Proof. Let the functions $\mu_s(\cdot)$ and $\sigma_s(\cdot)$ be as defined by Equations (11) and (12), respectively. At the observations ($x = x_s$ for $s \in \{1, \ldots, S\}$), the inequality $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ is satisfied, since $\mu_s(x_s|\mathcal{O}) + \sigma_s(x_s|\mathcal{O}) = \bar{f}(x_s|\mathcal{O}) = y_s$. Therefore, by letting $x_1 = 0$ and $x_S = 1$, only the closed intervals $x \in [x_{s-1}, x_s]$ for all $s \in \{2, \ldots, S\}$ need to be examined. The proof will focus on these closed intervals next.

Based on Equations (11) and (12) the inequality $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for the Dirichlet kernel at the closed intervals can be rewritten as

$$\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O}) \text{ where } x \in [x_{s-1}, x_s] \text{ for all } s \in \{2, \ldots, S\}. \tag{14}$$

For the benefit of the analysis, we normalize the function input for every interval by defining a normalized function argument as

$$\tau_s(x|\mathcal{O}) = \frac{x - x_{s-1}}{\Delta x_s}. \tag{15}$$

Where $\Delta x_s = x_s - x_{s-1}$ is the size of the interval, and $\tau_s(x|\mathcal{O}) \in [0, 1]$ for $x \in [x_{s-1}, x_s]$. All possible critical points of $\bar{f}(\cdot)$, $\hat{\tau}_s$, can be written as a function of $\nu_s(\mathcal{O})$ as

$$\hat{\tau}_s(\nu_s|\mathcal{O}) = \frac{1}{2} + \frac{(y_s - y_{s-1})}{2L_f \Delta x_s} \tag{16}$$

$$= \frac{1}{2}\left(1 + \nu_s(\mathcal{O})\right), \tag{17}$$

where $\nu_s(\mathcal{O}) \in [-1, 1]$ is defined as

$$\nu_s(\mathcal{O}) = \frac{y_s - y_{s-1}}{L_f \Delta x_s}.$$

Likewise, the upper bound function $\bar{f}(\cdot)$ (Definition 1.2) can be rewritten based on the normalized interval as

$$\bar{f}_s(x|\mathcal{O}) = \begin{cases} L_f \Delta x_s \tau_s(x|\mathcal{O}) + y_{s-1} & \text{if } 0 \leq \tau_s(x|\mathcal{O}) < \hat{\tau}_s(\nu_s|\mathcal{O}), \\ L_f \Delta x_s (1 - \tau_s(x|\mathcal{O})) + y_s & \text{if } \hat{\tau}_s(\nu_s|\mathcal{O}) \leq \tau_s(x|\mathcal{O}) \leq 1. \end{cases} \tag{18}$$

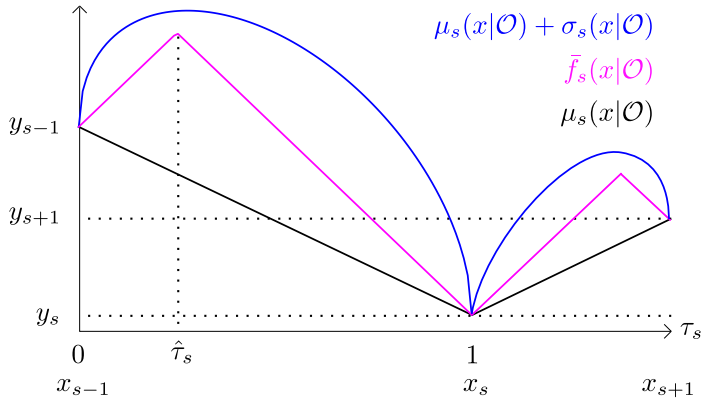


Figure 6: Graphical overview of the normalized function argument $\tau_s(x|\mathcal{O})$ over its domain $[0, 1]$. The domain corresponds to the interval $[x_{s-1}, x_{s+1}]$ where $\mu_s(\cdot)$ and $\sigma_s(\cdot)$ are based on the Dirichlet kernel and $\bar{f}_s(\cdot)$ is based on L_f and Δx_s. Based on the upper bound function, the domain can be divided into two intervals $[0, \hat{\tau}_s(\nu_s|\mathcal{O}))$ and $[\hat{\tau}_s(\nu_s|\mathcal{O}), 1]$.

A graphical overview of the normalized interval and the corresponding functions can be seen in Figure 6.

After defining these variables, two separate intervals can be considered, $[0, \hat{\tau}_s(\nu_s|\mathcal{O}))$ and $[\hat{\tau}_s(\nu_s|\mathcal{O}), 1]$. Both these intervals will be investigated next.

Interval $[0, \hat{\tau}_s(\nu_s|\mathcal{O}))$: Let $\tau_s(x|\mathcal{O}) \in [0, \hat{\tau}_s(\nu_s|\mathcal{O}))$. The mean function, given by Equation (11), on the interval can be rewritten based on the normalized function argument as

$$\begin{aligned}\mu_s(x|\mathcal{O}) &= \frac{y_{s-1}(x_s - x) + y_s(x - x_{s-1})}{x_s - x_{s-1}} \\ &= y_{s-1}(1 - \tau_s(x|\mathcal{O})) + y_s\tau_s(x|\mathcal{O}) \\ &= y_{s-1} + \tau_s(x|\mathcal{O})L_f\Delta x_s\nu_s(\mathcal{O}). \end{aligned} \tag{19}$$

Likewise, the variance function in Equation (12) can be rewritten as the deviation function as

$$\begin{aligned}\sigma_s(x|\mathcal{O}) &= \lambda\sqrt{\frac{-(x_s - x)(x_{s-1} - x)}{x_s - x_{s-1}}} \\ &= \lambda\sqrt{(1 - \tau_s(x|\mathcal{O}))\tau_s(x|\mathcal{O})\Delta x_s}. \end{aligned} \tag{20}$$

Substitution of Equations (18) to (20) into Equation (14) yields

$$\begin{aligned}&\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) \geq \bar{f}_s(x|\mathcal{O}) \\ &\qquad \lambda \geq (1 - \nu_s(\mathcal{O}))\sqrt{\frac{\tau_s(x|\mathcal{O})}{(1 - \tau_s(x|\mathcal{O}))}}L_f\sqrt{\Delta x_s}. \end{aligned} \tag{21}$$

Note that Equation (21) gives an explicit expression of the value for the kernel scale λ and all possible values of input/output pairs of the observations through the auxillary variable $\nu_s(\mathcal{O})$.

To analyze the values of λ for which the inequality of Equation (21) holds, the upper bound of the right-hand side is determined.

Since $\sqrt{\tau_s(x|\mathcal{O})/(1-\tau_s(x|\mathcal{O}))}$ is monotonically increasing with respect to $\tau_s(x|\mathcal{O})$, we find for $\tau_s(x|\mathcal{O})$ in the interval $[0, \hat{\tau}_s(\nu_s|\mathcal{O}))$,

$$\begin{aligned}(1-\nu_s(\mathcal{O}))\sqrt{\frac{\tau_s(x|\mathcal{O})}{(1-\tau_s(x|\mathcal{O}))}} &\leq (1-\nu_s(\mathcal{O}))\sqrt{\frac{\hat{\tau}_s(\nu_s|\mathcal{O})}{(1-\hat{\tau}_s(\nu_s|\mathcal{O}))}} \\ &\leq (1-\nu_s(\mathcal{O}))\sqrt{\frac{\frac{1}{2}(1+\nu_s(\mathcal{O}))}{(1-\frac{1}{2}(1+\nu_s(\mathcal{O})))}} \\ &\leq \sqrt{1-\nu_s(\mathcal{O})^2} \\ &\leq 1.\end{aligned}$$

Furthermore, since $\Delta x_s \leq \Delta x \leq 1$, we find through substitution of the upper bounds in Equation (21) that if $\lambda \geq L_f$, then Equation (21) is satisfied for all possible observations.

Interval $[\hat{\tau}_s(\nu_s|\mathcal{O}), 1]$: For this interval the same approach is applied. Let $\tau_s(x|\mathcal{O}) \in [\hat{\tau}_s(\nu_s|\mathcal{O}), 1]$; then substitution of Equations (18) to (20) in Equation (14) yields

$$\begin{aligned}\mu_s(x|\mathcal{O}) + \sigma_s(x|\mathcal{O}) &\geq \bar{f}_s(x|\mathcal{O}) \\ \lambda &\geq (1+\nu_s(\mathcal{O}))\sqrt{\frac{(1-\tau_s(x|\mathcal{O}))}{\tau_s(x|\mathcal{O})}}L_f\sqrt{\Delta x_s}. \qquad (22)\end{aligned}$$

Since $\sqrt{(1-\tau_s(x|\mathcal{O}))/\tau_s(x|\mathcal{O})}$ is monotonically decreasing with respect to $\tau_s(x|\mathcal{O})$, we find

$$(1+\nu_s(\mathcal{O}))\sqrt{\frac{(1-\tau_s(x|\mathcal{O}))}{\tau_s(x|\mathcal{O})}} \leq 1. \qquad (23)$$

Therefore, we conclude that for the interval $[\hat{\tau}_s(\nu_s|\mathcal{O}), 1]$ if $\lambda \geq L_f$ then Equation (22) is satisfied for all possible observations.

In conclusion, if $\lambda \geq L_f$, we find that the inequality of Equation (14) will hold for the intervals $[x_{s-1}, x_s]$ for $s \in \{1, \ldots, S\}$. Since $x_1 = 0$ and $x_S = 1$, Equation (13) hold for all $x \in (0, 1)$. Subsequently, $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ will holds for all $x \in [0, 1]$. □

According to Theorem 1.4, if $\lambda \geq L_f$ we find $\mu(x|\mathcal{O}) + \sigma(x|\mathcal{O}) \geq \bar{f}(x|\mathcal{O})$ for all $x \in \text{dom}(f)$. Applying Lemma 1.3 and setting $\xi = 0$, yields $\mathcal{U} \subseteq \mathcal{S}$ where $x^* \in \mathcal{S}$ for all observations. Subsequently, the Bayesian optimization will converge to the global optimum of the function. Note that for all functions without a normalized domain, the Lipschitz constant should be scaled according to the scaling required for the normalization of the domain.

6.2 Convergence of D-Bay Based on Global Optima of Utility Functions

As shown in Section 6.1, all agents can find the global optimum of the aggregate utility of their utility functions and the utility functions of their children through Bayesian optimization. In this section, it will be shown that given the global optima of the utility functions, D-Bay will find the global optimum of the objective function.

During the sample phase of D-Bay, none of the agents can optimize their variables without interaction with other agents. The interaction involves the sending of (top-down) **sample** messages $\mathfrak{S}$ and (bottom-up) **utility** messages $\mathfrak{U}$. Therefore, for the leaf agents, the optimization depends on their utility functions and the **sample** message of their parent, $\mathfrak{S}_j$, as

$$\hat{\rho}_{\mathbf{X}_i} = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\mathfrak{U}_i\right) = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\underset{f_n \in \mathbf{F}_i}{\eta}\left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j\right)\right) \qquad \forall a_i \;:\; \mathbf{C}_i = \emptyset. \tag{24}$$

When the kernel and acquisition function are selected as detailed in Section 6.1, the assignment $\hat{\rho}_{\mathbf{X}_i}$ is optimal with respect to the assignments of the (pseudo-)parents of agent a_i, since $\mathfrak{S}_j = \hat{\rho}_{\mathbf{P}_i} \cup \hat{\rho}_{\mathbf{PP}_i}$. Consequently, the optimal assignment results in the optimal value for the **utility** message $\mathfrak{U}_i^j$ given the **sample** message $\mathfrak{S}_j$.

This optimal value is sent as a **utility** message to the parents of the leaf agents and results in the following assignment for the other agents:

$$\hat{\rho}_{\mathbf{X}_i} = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\mathfrak{U}_i, \hat{\mathfrak{U}}_i\right) = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}}{\arg\min}\, \eta\left(\underset{f_n \in \mathbf{F}_i}{\eta}\left(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j\right), \hat{\mathfrak{U}}_i\right) \qquad \forall a_i \;:\; \mathbf{C}_i \neq \emptyset. \tag{25}$$

The aggregated **utility** message $\hat{\mathfrak{U}}_i = \underset{a_k \in \mathbf{C}_i}{\eta}\left(\mathfrak{U}_k^i\right)$ is the aggregate of the optimal **utility** messages of all children given an assignment of a_i. Therefore, agent a_i can calculate the optimal assignment with regard to its parent **sample** message.

This process is repeated until the root agent a_1 has received all **utility** messages from its children. As the root agent does not have any (pseudo-)parents, Equation (25) can be rewritten as

$$\hat{\rho}_{\mathbf{X}_1} = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_1}}{\arg\min}\, \eta\left(\mathfrak{U}_1, \hat{\mathfrak{U}}_1\right) = \underset{\rho \in \mathbf{\Sigma}_{\mathbf{X}_1}}{\arg\min}\, \eta\left(\underset{f_n \in \mathbf{F}_1}{\eta}\left(f_n(\rho_{\mathbf{V}_n})\right), \hat{\mathfrak{U}}_1\right) = \rho^*_{\mathbf{X}_1}. \tag{26}$$

Note that $\hat{\mathfrak{U}}_1$ holds the aggregate utility value of all other agents based on the sample of the root agent. For that reason, if the root agent finds the optimal assignment $\hat{\rho}_{\mathbf{X}_1}$ it is equal to the optimum of the objective function $\rho^*_{\mathbf{X}_1}$.

After the root agent has found the optimal assignment of its variables it starts the final phase of D-Bay. In this phase the root agent sends the optimal assignment as a **final** message to its children, $\hat{\mathfrak{S}}_1 = \{\rho^*_{\mathbf{X}_1}\}$. Based on that optimal sample all agents are able to determine their optimal assignments, as shown in Equation (25), and append their optimal assignment to the final message before sending the final message to their children, i.e. $\hat{\mathfrak{S}}_i = \{\rho^*_{\mathbf{X}_i}\} \cup \hat{\mathfrak{S}}_j$. This process is repeated until the leaf agents are reached and all agents have assigned the globally optimal values to their variables.

6.3 Summary

In Section 6.1 it was shown that, based on the Lipschitz constant of a utility function (or aggregate of functions), the kernel and acquisition function (and their parameters) can be appropriately selected to guarantee convergence to the global optimum of the utility function. Subsequently, Section 6.2 has shown that, if the agents can find the global optimum of the aggregate of the utility functions, D-Bay will converge to the global optimum of the objective function. Combining these results proves the convergence of D-Bay to the global optimum of the objective function for utility functions with known Lipschitz constants.

7. Simulation Results

In this section, the performance of the D-Bay algorithm is compared to DCOP solvers and C-DCOP solvers. The sampling-based DCOP solvers Sequential Distributed Gibbs (SD-Gibbs) (Nguyen et al., 2019) and DUCT (Ottens et al., 2017) are chosen to compare the performance of the D-Bay algorithm with discrete solvers of the same class. Additionally, DPOP (Petcu & Faltings, 2005) is added to the comparison to represent the optimal solution of the DCOP solvers, since these solvers operate on the same domains and DPOP is a complete solver. The C-DCOP solvers AC-DPOP (Hoang et al., 2020) and PFD (Choudhury et al., 2020) are selected as both achieve higher performance than the HCMS algorithm of Voice et al. (2010).

The implementation of DPOP is included within the pyDCOP library (Rust, Picard, & Ramparany, 2019). All other algorithms have been included in the pyDCOP library and made available publicly[1]. The simulations are conducted on a 2.1 GHz Intel Xeon Gold 6152 CPU machine with sufficient memory for the requirements of all solvers and the computation time is limited to one hour. All algorithms are evaluated on two types of problems: random graphs and sensor coordination problems.

The hyperparameters of the solvers are fixed for all experiments and their values are listed in Table 1. If available, the values are taken equal to the listed values in the original works. The termination criteria for the incomplete solvers will be related to the parameters of the experiments.

Table 1: Hyperparameters of DCOP solvers used during simulations.

Algorithm	Hyperparameters
DPOP	-
SD-Gibbs	iterations $= 20$
DUCT	$\epsilon = 0.6, \delta = 0.1$
AC-DPOP	iterations $= 100, \delta = 0.001, \alpha = 0.01$
PFD	particles $= 2000, w = 0.9, c_1 = 0.9, c_2 = 0.1, \max_{f_c} = 5, \max_{s_c} = 15$
D-Bay	$\lambda = L_f$

The experiments are defined based on three parameters: number of constraint checks, number of agents $|\mathbf{A}|$, and density of the graph p_1. The density of the graph is defined as the

1. https://gitlab.com/jfransman/pyDcop/

ratio between the number of edges and the maximal number of possible edges. The number of constraint checks is used as a parameter to compare the DCOP and C-DCOP solvers based on computational efficiency. The efficiency of the solver is an important measure for problems for which the utility functions are computationally expensive to evaluate. Note that when comparing solvers for problems in which communication is the bottleneck non-concurrent constraint checks (Meisels, Kaplansky, Razgon, & Zivan, 2002) are more suitable.

For all DCOP solvers, the continuous domains are uniformly discretized in a preprocessing step to convert the C-DCOP into a DCOP. The domain cardinality of the generated DCOPs is related to the constraint checks during the solving procedure. More values within a domain will constitute more possible evaluations of the utility functions. The C-DCOP solvers operate directly on the continuous domains; however, most solvers have a parameter that is analogous to the domain cardinality. The AC-DPOP algorithm requires the discretization of the domains within a preprocessing step. As Hoang et al. (2020) provides no method for defining the optimal level of discretization, the discretization is set equal to that of the DCOP. The PFD algorithm initiates by selecting a random value for every particle using a uniform distribution from the domains and updates the values during every iteration. The D-Bay algorithm samples the continuous domains dynamically without discretization. For that reason, the *number of samples* parameter is used as a threshold during the experiments.

7.1 Random Graphs

For the generation of the random graph experiments, the NetworkX (Hagberg, Schult, & Swart, 2020) generator, embedded within the pyDCOP library, is used. Based on the randomly created graph, a C-DCOP is generated by allocating a variable (and agent) to every node and defining utility functions for all edges. The bi-modal Bird function (Mishra, 2006) (shown in Figure 7) is used as utility function. The Bird function was created as a test function for global optimization and contains multiple local optima at different values.

Based on the three defined parameters, numerous experiments are conducted; the number of agents is varied from 3 to 10, and the graph density is varied from 0.1 to 0.4 with increments of 0.02. All experiments are repeated 50 times and the median of the most illustrious results are shown in Figure 8.

In Figures 8a and 8b the relative utility is calculated based on the utility found by a centralized exhaustive search-based algorithm on densely discretized domains. The relative utility shows several important properties of the solvers. In Figure 8a all solvers show an increase of relative utility that converges when the number of constraint checks is increased. The performance of the C-DCOP solvers differs significantly. The AC-DPOP solver shows a nearly constant performance but requires a large number of constraint checks. This is expected as the AC-DPOP solver initially starts with domain values that are equal to that of the discrete solvers and then the agents update their values based on a local gradient descent method. Upon further investigating the cause of the constant performance, we found that most of the domain values converge to the same local optima. This effectively reduces the number of domain values that are evaluated. During optimization, the utility values at the local optima of the Bird function are sent to the parents of the agents after which these

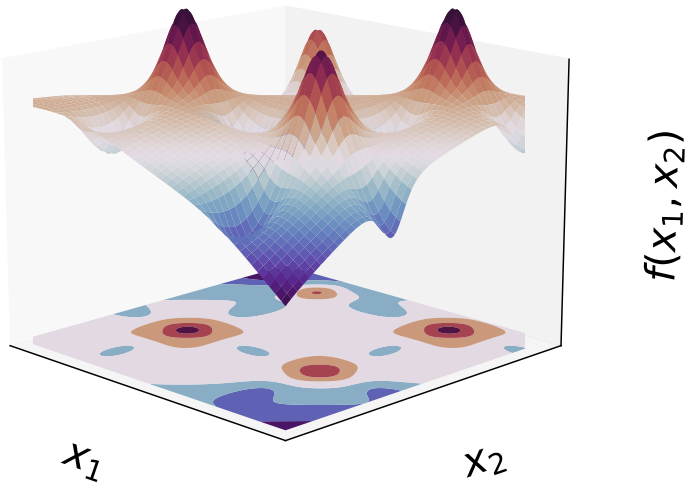


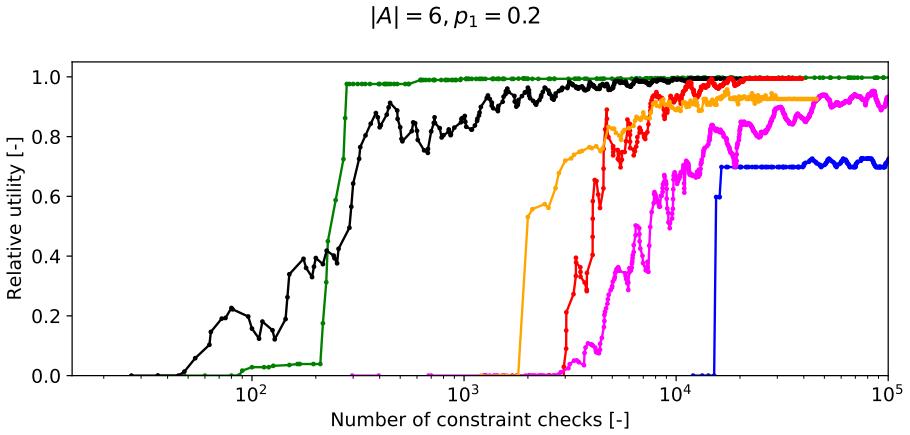
Figure 7: Combined view of the function values and a contour plot of the Bird function (Mishra, 2006) on the domain $x_i \in [-2\pi, 2\pi]$ for $i = 1, 2$.

values are interpolated. This (linear) interpolation results in a large overestimation of the utility values between the optima, therefore the solver does not escape the local optima and the performance does not improve when the constraint checks in increased.

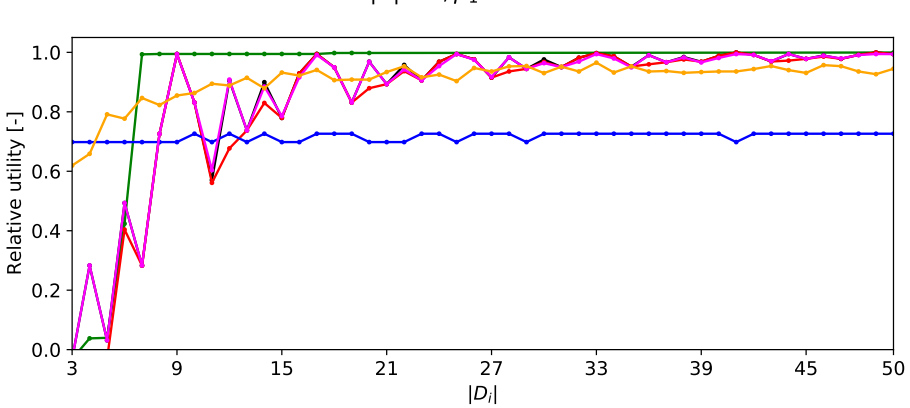
The performance of the PFD solver steadily increases for larger numbers of constraint checks. This behavior is expected since the domain values are updated during optimization based on the shared particles. The particles represent (partial) allocations of the variables within the C-DCOP. The relatively large number of particles (2000) increases the chance of producing particles that represent high utility allocations. These particles influence other particles during their value updates, thereby improving performance. This effect also holds for particles that are initialized near local optima. These particles influence other particles by driving them away from regions with high global utility.

Compared to other solvers, D-Bay shows the most consistent performance, in terms of achieved relative utility. It shows a (near) monotonic increase in performance. This can be explained by the property of the Bayesian optimization. Bayesian optimization combined with the appropriate choice of the kernel based on the Lipschitz constant ensures that all samples are selected such that the largest amount of information about the optima is gained. This allows for the exclusion of large regions of the domains, which effectively focuses the sampling on high-utility areas of the search space.

The results of the DCOP solvers (DPOP, SD-Gibbs, DUCT) show increasing utility that approaches the optimum. Compared to the C-DCOP solvers, the utility does not increase as smoothly when the number of constraint checks is increased. Upon further investigation, the performance of the DCOP solvers shows a clear dependency on the selection of the domain values. This is visible by the irregular increase of the relative utility for increasing domain cardinality as shown in Figure 8b. For a domain cardinality of 9, the discretization resulted in an excellent performance. This highlights a drawback of solvers that are dependent on discretization, as increasing the cardinality will not always guarantee better performance. In other words, there is no monotonic relation between the cardinality of



(a) Relative utility for varying numbers of constraint checks.



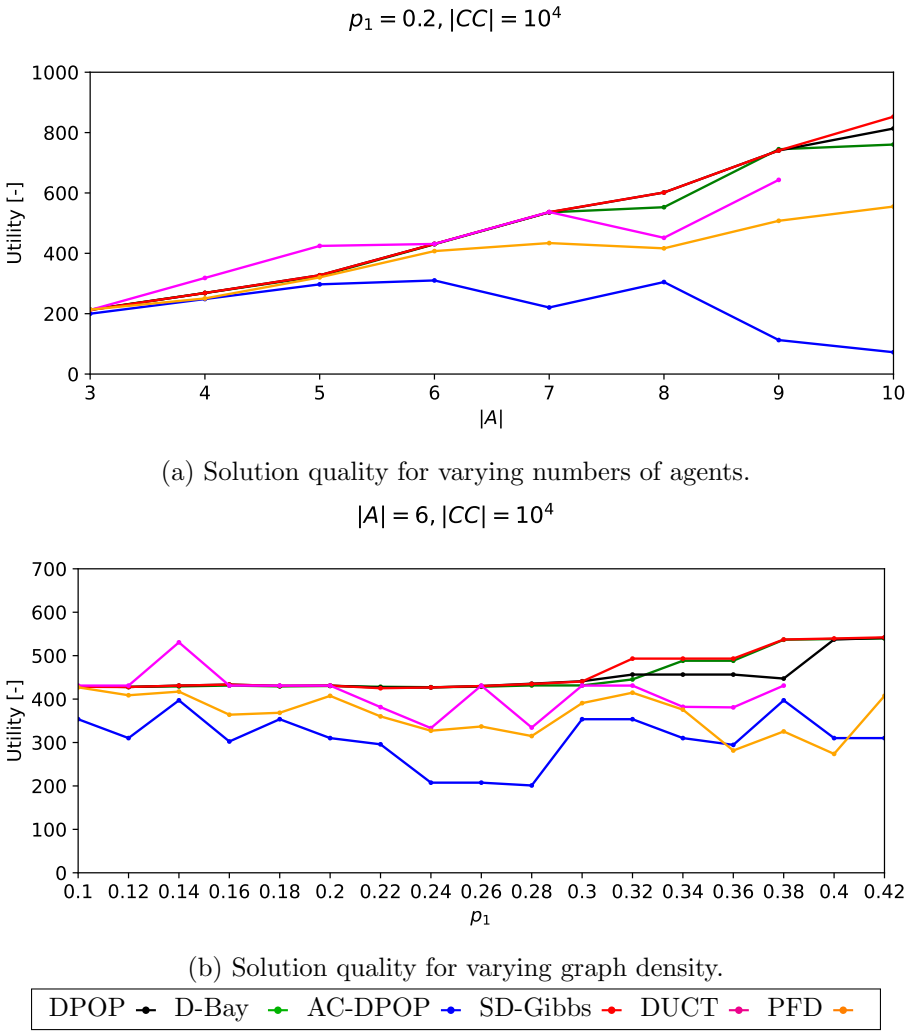
(b) Relative utility for varying domain cardinality $|D_i|$.

DPOP D-Bay AC-DPOP SD-Gibbs DUCT PFD

Figure 8: Experimental results for the random graph problems.

the domains and the performance. The results of the DCOP solvers (DPOP, SD-Gibbs, DUCT) can be seen to overlap significantly. The DPOP algorithm yields the optimal solution of the discretized C-DCOP. This indicates that both SD-Gibbs and DUCT show close-to-optimal performance with high consistency. DUCT outperforms SD-Gibbs for low values of the domain cardinality however, for all higher values, SD-Gibbs achieves close to optimal performance. However, these algorithms require more constraint checks as can be seen in Figure 8a.

The performance of the D-Bay algorithm is very close to the optimum for 5×10^2 constraint checks. Similar performance is consistently achieved by the DCOP solvers at 1×10^4 constraint checks. To show D-Bay has a high sample efficiency independent of the number of agents and the density of the graph, these values are used in the comparison for the experiments with a varying number of agents and graph density. The results are shown in Figure 9.



(a) Solution quality for varying numbers of agents.

(b) Solution quality for varying graph density.

Figure 9: Comparison of performance based on a varying number of agents and graph density.

Figure 9a shows the results for problems with a graph density of 0.2, where the number of agents is varied from 3 to 10. In Figure 9b the results for a 6 agent problem are shown, where the graph density is varied from 0.1 to 0.4 with increments of 0.02. Within both figures, the performance for all solvers is shown to be closely related to the results from Figure 8b. This demonstrates the fact that the sample efficiency of D-Bay holds for numerous random graphs. In other words, D-Bay achieves high performance with a limited number of samples and this performance is achieved by the compared algorithms only for a larger number of (evaluated) domain values.

7.2 Sensor Coordination Problems

The sensor coordination problem is an optimization problem in which every agent needs to orient its sensor to observe targets as accurately as possible. A real-world analogy would be the optimization of the orientation of multiple cameras based on image recognition. This problem is modeled within the DCOP framework as a distributed problem. The image recognition process can require significant computational effort depending on the image quality and the type of target(s). It is therefore computationally intensive to check every orientation of the camera, especially for a centralized approach. Even for a relatively small number of agents, a DCOP representation of these problems will result in a large search space.

Within the sensor coordination problem, all sensors are modeled identically in terms of their sensor range l and angle of view β. These properties, combined with the position of the sensor, determine the observation domain of the sensor. This domain defines all locations that could be observed by the sensor. The orientation ω_i of the sensor of agent a_i determines the observed area within the observation domain. A target is detected when it is located within this area. For every detected target, a (positive) utility value is allocated to the agent. The maximum utility is allocated when the sensor is oriented directly at the target. The utility value decreases linearly towards the edges of the observation area. The optimal utility is determined by the optimal solution of the centralized optimization approach with 720 samples (0.5 degree resolution) for every domain. The parameters of the problem are the number of targets T, the number of sensors N, the sensor range l, the angle of view β of the sensors, and the arrangement of the sensors. The sensors are arranged in an equally distanced rectangular grid. Various configurations are simulated, where a configuration indicates the number of rows and columns of the grid. The sensors are positioned such that the combined observation domains of all sensors are maximized without allowing unobservable areas between the sensors. For this reason, the distance between the sensors of the same row or column is $\sqrt{2}l$. The locations of the targets t are uniformly distributed within the combined observation domains of the sensors. In the experiments, the problems are generated with 6 sensors, 12 targets, and with identical sensor properties, where the sensor range is set to $l = 1$ and the angle of view is set to $\beta = 36°$. A graphical example of the simulated sensor coordination problems can be seen in Figure 10. The sensor coordination problem is described within the C-DCOP framework as follows:

- $\mathbf{A} = \{a_1, \ldots, a_M\}$ is the set of agents, where M is the number of agents. The position of agent i is denoted as $p_i \in \mathbb{R}^2$.
- $\mathbf{X} = \{\omega_1, \ldots, \omega_N\}$ is the set of sensor orientations, where $N = M$.
- $\mathbf{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_N\}$, where $\mathbf{D}_i = (-180°, 180°)$ for all $i = 1, \ldots, N$ indicating all possible values of sensor orientation ω_i.
- $\mathbf{F} = \{f_n\}_{n=1}^{T}$ is the set of utility functions associated with the observation of the targets. The number of targets is denoted by $T \in \mathbb{N}$. Target n is located at position $t_n \in \mathbb{R}^2$. The utility functions of the targets are described as $f_n = \max_{i=1,\ldots,N}(f_{n,i})$

for $n = 1, \ldots, T$, where

$$f_{n,i} = \begin{cases} 1 - |\omega_i - \angle\overrightarrow{p_i t_n}|/\beta & \text{if } \|\overrightarrow{p_i t_n}\| \leq l \text{ and } |\omega_i - \angle\overrightarrow{p_i t_n}| \leq \beta \\ 0 & \text{otherwise} \end{cases}$$

and $\overrightarrow{p_i t_n}$ denotes the vector between the location of the target t_n and the position of the agent a_i. Figure 11 shows an example of the utility value as a function of the angle of view.

- $\alpha(\omega_i) = a_i$ for $i = 1, \ldots, N$ allocating a single sensor to every agent.
- $\eta = \sum(\cdot)$, resulting in the goal function $G(\cdot) = \sum_{f_n \in F} f_n(\cdot)$.

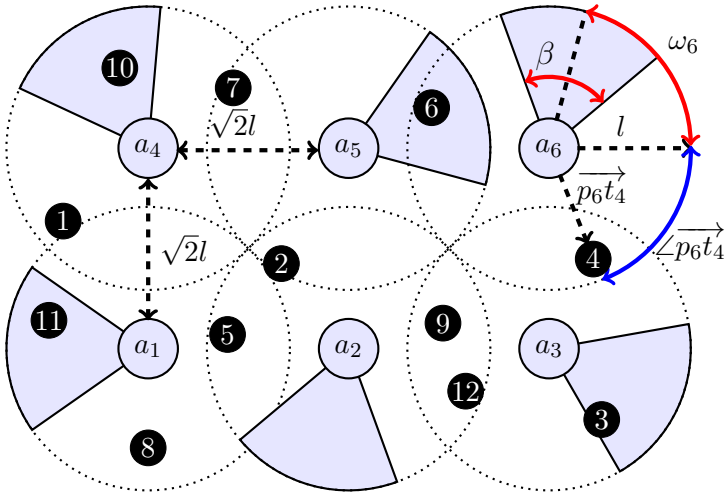


Figure 10: Graphical example of a sensor coordination problem with 6 sensors and 12 targets. The sensors a_i are arranged in an equally distanced rectangular grid. The distance between the sensors is based on the sensor range l. The observation domain is indicated by a dotted circle centered around the position of the sensor. The observed area of the sensors is shown as shaded areas and is based on the angle of view β and the orientation ω_i. The targets are shown as annotated black circles.

In this section, the performance of D-Bay is empirically evaluated using the achieved relative utility as a function of the number of samples. This metric is important to consider if the evaluation of the utility functions by the agents is computationally expensive. The relative utility allows for the comparison of the results over various randomly generated problems. The achieved relative utility is defined as the achieved utility divided by the optimal utility generated by centralized exhaustive search based on densely discretized domains. The number of samples is defined as the maximum number of domain values checked by an agent and allows for a comparison based on sample efficiency per agent instead of the algorithm as a whole. For the DCOP solvers, the number of samples (per agent) is equal to the domain cardinality of the variables of the discretized C-DCOPs. The

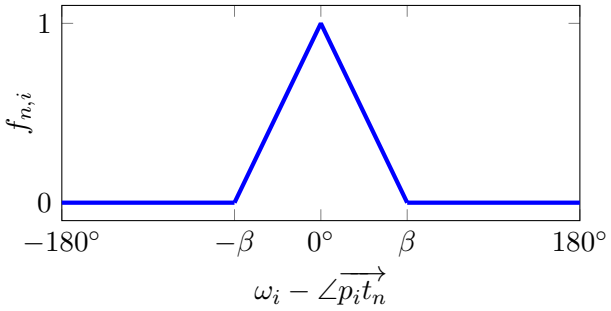


Figure 11: Utility function $f_{n,i}$ indicates the utility of agent a_i for the observation of target t_n if the target is within the sensor range l ($\|\overrightarrow{p_i t_n}\| \leq l$). The angle of view of the sensor is denoted as β. On the horizontal axis the difference between the sensor orientation ω_i and the angle between the position of the agent and the position of the target $\angle\overrightarrow{p_i t_n}$ is given.

C-DCOP solvers iteratively update the domain values by either local gradient descent (AC-DPOP), particle velocity update (PFD), or sampling (D-Bay). In order to compare these solvers, the number of updates of the value of the root agent is used.

The performance results of D-Bay compared to the DCOP and C-DCOP solvers are given in Figure 12 for various configurations of the sensor coordination problem. Similar to the results of D-Bay in Figure 8a, the sample efficiency of D-Bay enables it to outperform both the DCOP solvers as well as the C-DCOP solvers. The performance of PFD slightly surpasses D-Bay for the 1x2 configuration but falls behind all other solvers for the 2x2, 2x3, and 3x2 configurations. This indicates that the performance of PFD largely depends on the random initialization of the particles.

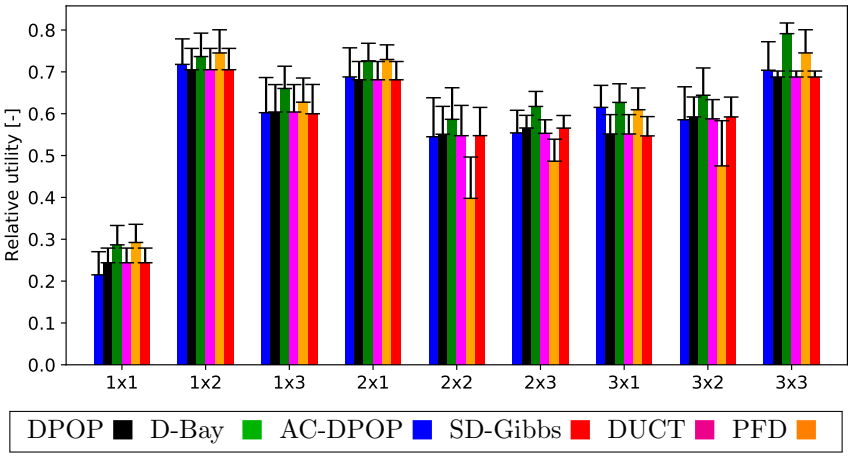
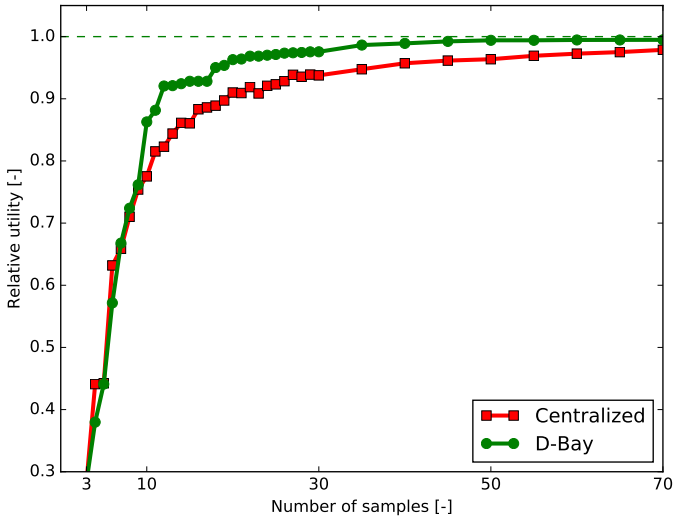


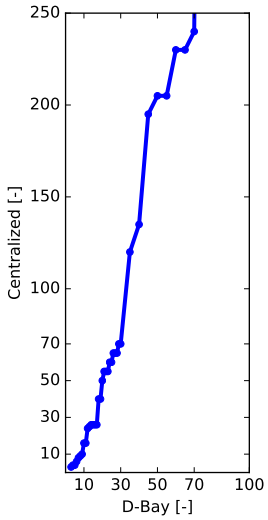
Figure 12: Sensor configurations for 10 targets and 10 number of samples.

The performance results of D-Bay compared to the centralized approach are presented in Figure 13. This figure shows the results for 30 randomly generated problems for 6 sensors and 12 targets. The results show an increase in the achieved relative utility of D-Bay

compared to the centralized approach based on the number of samples. The difference in achieved utility can be explained by investigating the sampling strategies. The centralized approach samples the sensor orientations equidistantly. Therefore, as the number of samples is increased, the resolution of the samples increases uniformly for the centralized approach. D-Bay samples dynamically to balance exploration and exploitation based on all previously acquired observations. Consequently, D-Bay will initially focus on exploration and eventually focus on exploitation. This behavior is clearly visible in Figure 13a in the range between samples 3 and 10. Within this range, D-Bay samples the sensor orientations equidistantly focussing on exploration. The sampling behavior is identical to the centralized approach, which can be seen in the similarity in achieved utility. For more than 11 number of samples, the achieved utility of D-Bay increases substantially. This can be explained based on the angle of view of 36° of the sensors during the experiments. At 10 samples the entire observation domain of a sensor is observed. Afterward, the switch to the exploitation of the observations increases the achieved utility more than the continued exploration of the centralized approach. The advantage is even more prominent when comparing the number of samples required by the centralized approach to achieve equal utility to D-Bay, as shown in Figure 13b. This clearly shows the advantage of the dynamic sampling of D-Bay over equidistant sampling.



(a) Achieved relative utility.



(b) Required number of samples for equal utility.

Figure 13: Simulation results for randomly generated sensor coordination problems with 6 sensors and 12 targets. The figures show the average result of 30 randomly generated problems. Figure 13a shows the achieved utility of D-Bay and the centralized approach relative to the optimum. Note that the achieved utility for both algorithms is equal for 3 samples since the first 3 samples are equidistantly spaced for D-Bay. Figure 13b shows the number of samples required by the centralized approach to achieve the same utility as D-Bay.

8. Conclusions

In this paper, the novel algorithm called Distributed Bayesian (D-Bay) has been introduced to solve Continuous Distributed Constraint Optimization Problems (C-DCOPs). Within D-Bay, the continuous domains are sampled based on Bayesian optimization. This removes the need for the discretization of the domains and balances exploration and exploitation of the global search space by incorporating knowledge about the utility functions within the kernels of the probabilistic models. Compared to DCOP solvers, which require discretization of the C-DCOP, it results in a reduction of the computational and memory demands of the individual agents. For utility functions with known Lipschitz constants, D-Bay is proven to converge to the global optimum solution of the C-DCOP.

Random graphs and sensor coordination problems have been used to evaluate the performance of D-Bay. The results show that D-Bay outperforms a centralized approach as well as state-of-the-art DCOP and C-DCOP solvers based on the achieved utility as a function of the required number of samples. This sample efficiency is a result of the application of Bayesian optimization with the D-Bay algorithm. Implementations of the proposed algorithm and the state-of-the-art DCOP and C-DCOP solver have been added to the open-source software library pyDCOP (Rust et al., 2019) and made available publicly[2].

In future work, D-Bay will be extended towards dynamic DCOPs (Fioretto et al., 2018) in which the agents need to optimize a dynamic problem at every time step. An extension will increase the applicability of the proposed algorithm to dynamic mobile sensor platform problems in which tracking of targets is an important factor, such as multi-agent surveillance. In a dynamic adaptation of the sensor coordination problem, the locations of the targets change over time based on the target properties, such as velocity and turn radius.

Acknowledgments

The authors would like to thank dr. Peyman Mohajerin Esfahani of the Delft Center for Systems and Control at the Delft University of Technology for his insightful comments.

2. `https://gitlab.com/jfransman/pyDcop/`

Appendix A. The Distributed Bayesian Algorithm

In this appendix, the formal description of the distributed Bayesian algorithm is presented.

Algorithm 2: Distributed Bayesian (D-Bay) for agent a_i

Input : $\mathbf{P}_i, \mathbf{PP}_i, \mathbf{C}_i, \mathbf{PC}_i, \mathbf{F}_{a_i}, \mathbf{F}_{\mathbf{P}_i}, \mathbf{X}_i, \kappa$
Output: $\hat{\rho}_{\mathbf{X}_i}$
Initialization
  if root agent **then**
    while *not threshold reached* **do**
      $\mathfrak{U}_i^j \coloneqq$ `optimizeLocalVariables(`$\emptyset$`)`;
    end
    `processFinal(`$\emptyset$`)`;
when received sample $\mathfrak{S}_j$ *from parent* $\mathbf{P}_i$
  while *not threshold reached* **do**
    $\mathfrak{U}_i^j \coloneqq$ `optimizeLocalVariables(`$\mathfrak{S}_j$`)`;
  end
  `send(`$\mathbf{P}_i$`,` $\mathfrak{U}_i^j$`)`;
when received final $\hat{\mathfrak{S}}_j$ *from parent* $\mathbf{P}_i$
  `processFinal(`$\hat{\mathfrak{S}}_j$`)`;

Function `optimizeLocalVariables(`$\mathfrak{S}_j$`)`
  $\rho_{\mathbf{X}_i} \coloneqq$ `computeOptimalSample(`κ`)`;
  $\mathfrak{S}_i \coloneqq \mathfrak{S}_j \cup \{\rho_{\mathbf{X}_i}\}$;
  $\mathfrak{U}_i^j \coloneqq$ `calculateUtility(`$\mathfrak{S}_i$`)`;
  return $\mathfrak{U}_i^j$;

Function `calculateUtility(`$\mathfrak{S}_i$`)`
  $\mathfrak{U}_i \coloneqq \min_{\rho \in \mathbf{\Sigma}_{\mathbf{X}_i}} \eta_{f_n \in \mathbf{F}_i} \Big(f_n(\rho_{\mathbf{V}_n} \mid \mathfrak{S}_j) \Big)$;
  if $\mathbf{C}_i \neq \emptyset$ **then**
    $\hat{\mathfrak{U}}_i \coloneqq$ `getChildUtility(`$\mathfrak{S}_i$`)`;
    $\mathfrak{U}_i^j \coloneqq \eta \left(\mathfrak{U}_i, \hat{\mathfrak{U}}_i \right)$;
  else
    $\mathfrak{U}_i^j \coloneqq \mathfrak{U}_i$;
  `storeUtility(`$\mathfrak{U}_i^j$`,` $\mathfrak{S}_i$`)`;
  return $\mathfrak{U}_i^j$;

Function `getChildUtility`($\mathfrak{S}_i$)
  foreach $a_k \in \mathbf{C}_i$ **do** `send`(a_k, $\mathfrak{S}_i$);
  when received $\mathfrak{U}_k^i$ *from all* $a_k \in \mathbf{C}_i$
    $\hat{\mathfrak{U}}_i \coloneqq \underset{a_k \in \mathbf{C}_i}{\eta} \left(\mathfrak{U}_k^i\right)$;
  return $\hat{\mathfrak{U}}_i$;

Procedure `processFinal`($\hat{\mathfrak{S}}_j$)
  $\hat{\rho}_{\mathbf{X}_i} \coloneqq$ `retrieveOptimalLocalSample`($\hat{\mathfrak{S}}_j$);
  $\hat{\mathfrak{S}}_i \coloneqq \hat{\mathfrak{S}}_j \cup \{\hat{\rho}_{\mathbf{X}_i}\}$;
  foreach $a_k \in \mathbf{C}_i$ **do** `send`(a_k, $\hat{\mathfrak{S}}_i$);

Appendix B. Dirichlet Kernel Interval Functions

In this appendix, the derivation of the mean and variance function corresponding to the Dirichlet kernel are presented.

As shown in the work of Ding and Zhang (2018, Theorem 2), a kernel κ of the Markovian class reduces the mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot|\mathcal{O})$ of the posterior on the interval between observations as given in Equations (8) and (9), respectively. For the Dirichlet kernel as defined by Equation (10), for a normalized domain $x_i, x_j \in [0, 1]$ and the kernel scale parameter λ, the non-zero elements of the $\boldsymbol{K}_s^{-1}(\mathcal{O})$ matrix are given by

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s} = \begin{cases} \lambda^{-2} \frac{x_1}{x_1\left(x_2 - x_1\right)}, & \text{if } s = 1, \\ \lambda^{-2} \frac{(x_{s+1} - x_{s-1})}{(x_s - x_{s-1})(x_{s+1} - x_s)}, & \text{if } s \in \{2, \ldots, S-1\}, \\ \lambda^{-2} \frac{(1 - x_{S-1})}{(1 - x_S)(x_S - x_{S-1})}, & \text{if } s = S, \end{cases}$$

and

$$(\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s-1,s} = (\boldsymbol{K}_s^{-1}(\mathcal{O}))_{s,s-1} = \frac{-\lambda^{-2}}{(x_s - x_{s-1})}, \; s = 2, \ldots, S.$$

The mean function $\mu_s(\cdot)$ and the variance function $\sigma_s^2(\cdot|\mathcal{O})$ for the Dirichlet kernel can be rewritten accordingly as

$$\begin{aligned}
\mu_s(x|\mathcal{O}) &= \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathcal{O})\boldsymbol{K}_s^{-1}(\mathcal{O})\boldsymbol{y}_s(\mathcal{O}) \\
&= \begin{bmatrix} 0 & \dots & 0 & \frac{x_s - x}{x_s - x_{s-1}} & \frac{x - x_{s-1}}{x_s - x_{s-1}} & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_{s-2} \\ y_{s-1} \\ y_s \\ y_{s+1} \\ \vdots \\ y_S \end{bmatrix} \\
&= \begin{bmatrix} \frac{x_s - x}{x_s - x_{s-1}} & \frac{x - x_{s-1}}{x_s - x_{s-1}} \end{bmatrix} \begin{bmatrix} y_{s-1} \\ y_s \end{bmatrix} \\
&= \frac{y_{s-1}(x_s - x) + y_s(x - x_{s-1})}{x_s - x_{s-1}}
\end{aligned} \tag{27}$$

and

$$\begin{aligned}
\sigma_s^2(x|\mathcal{O}) &= \kappa(x,x) - \boldsymbol{\kappa}_s^{\mathrm{T}}(x,\mathcal{O})\boldsymbol{K}_s^{-1}(\mathcal{O})\boldsymbol{\kappa}_s(x,\mathcal{O}) \\
&= \lambda^2 x(1-x) - \begin{bmatrix} 0 & \dots & 0 & \frac{x_s - x}{x_s - x_{s-1}} & \frac{x - x_{s-1}}{x_s - x_{s-1}} & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \lambda^2 x_1(1-x) \\ \vdots \\ \lambda^2 x_{s-2}(1-x) \\ \lambda^2 x_{s-1}(1-x) \\ \lambda^2 x(1-x_s) \\ \lambda^2 x(1-x_{s+1}) \\ \vdots \\ \lambda^2 x(1-x_S) \end{bmatrix} \\
&= \lambda^2 \left(x(1-x) - \left(\frac{x_{s-1}(1-x)(x_s - x)}{x_s - x_{s-1}} + \frac{x(1-x_s)(x - x_{s-1})}{x_s - x_{s-1}} \right) \right) \\
&= \lambda^2 \frac{-(x_s - x)(x_{s-1} - x)}{x_s - x_{s-1}}.
\end{aligned} \tag{28}$$

References

Acevedo, J. J., Arrue, B. C., Maza, I., & Ollero, A. (2013). Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions. *Journal of Intelligent and Robotic Systems: Theory and Applications*, *70*, 329–345.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2-3), 235–256.

Awerbuch, B. (1985). A new distributed depth-first-search algorithm. *Information Processing Letters*, *20*(3), 147–150.

Barbosa, V. C. (1996). *An introduction to distributed algorithms*. Mit Press.

Bather, J. A., Berry, D. A., & Fristedt, B. (1986). *Bandit Problems: Sequential Allocation of Experiments.*, Vol. 149.

Brochu, E., Cora, V. M., & de Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv*.

Bubeck, S., Munos, R., Stoltz, G., & Szepesvári, C. (2011). X-armed bandits. *Journal of Machine Learning Research (JMLR)*, *12*(5), 1655–1695.

Cerquides, J., Farinelli, A., Meseguer, P., & Ramchurn, S. D. (2014). A tutorial on optimization for multi-agent systems. *The Computer Journal*, *57*(6), 799–824.

Chechetka, A., & Sycara, K. (2005). A decentralized variable ordering method for distributed constraint optimization. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1307–1308.

Chen, Z., He, Z., & He, C. (2017). An improved DPOP algorithm based on breadth first search pseudo-tree for distributed constraint optimization. *Applied Intelligence*, *47*(3), 607–623.

Choudhury, M., Mahmud, S., & Khan, M. M. (2020). A particle swarm based algorithm for functional distributed constraint optimization problems.. Vol. 34, pp. 7111–7118.

Ding, L., & Zhang, X. (2018). Scalable stochastic kriging with Markovian covariances. *arXiv*.

Duvenaud, D. K., Nickisch, H., & Rasmussen, C. E. (2011). Additive Gaussian processes. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 226–234.

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

Freuder, E. C., & Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1076–1078.

Gallager, R. G., Humblet, P. a., & Spira, P. M. (1983). A distributed algorithm for minimum-weight spanning trees. *Transactions on Programming Languages and Systems (TOPLAS)*, *5*(1), 66–77.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Transactions on Pattern Analysis and Machine Intelligence*, pp. 721–741.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research (JAIR)*, *34*, 61–88.

Hagberg, A., Schult, D., & Swart, P. (2020). Networkx..

Hamadi, Y., & Quinqueton, J. (1998). Backtracking in distributed constraint networks. In *European Association for Artificial Intelligence (ECAI)*, pp. 219–223.

Hoang, K. D., Yeoh, W., Yokoo, M., & Rabinovich, Z. (2020). New algorithms for continuous distributed constraint optimization problems. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 502–510.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *International Conference on Neural Networks (IJCNN)*, Vol. 4, pp. 1942–1948.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pp. 282–293.

Kushner, H. J. (1964). A new method for locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, *86*(1), 97–106.

Leite, A. R., Enembreck, F., & Barthès, J.-P. A. (2014). Distributed constraint optimization problems: review and perspectives. *Expert Systems with Applications*, *41*(11), 5139–5157.

Lizotte, D. J., Greiner, R., & Schuurmans, D. (2012). An experimental methodology for response surface optimization methods. *Journal of Global Optimization*, *53*(4), 699–736.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation*, *4*(3), 415–447.

MacKay, D. J. (1994). Bayesian nonlinear modeling for the prediction competition. *American Society of Heating, Refrigerating and Air-Conditioning Engineers Transactions*, *100*(2), 1053–1062.

Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., & Varakantham, P. (2004). Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Meisels, A., Kaplansky, E., Razgon, I., & Zivan, R. (2002). Comparing performance of distributed constraints processing algorithms. In *AAMAS workshop on Distributed Constraint Reasoning (DCR)*, pp. 86–93.

Meisels, A. (2007). *Distributed search by constrained agents: algorithms, performance, communication*. Springer Science & Business Media.

Micchelli, C. A., Xu, Y., & Zhang, H. (2006). Universal kernels. *Journal of Machine Learning Research (JMLR)*, *7*, 2651–2667.

Minasny, B., & McBratney, A. B. (2005). The matérn function as a general model for soil variograms. *Geoderma*, *128*, 192–207.

Mishra, S. K. (2006). Some new test functions for global optimization and performance of repulsive particle swarm method. Tech. rep..

Mockus, J. (1982). The Bayesian approach to global optimization. *System Modeling and Optimization*, *38*, 473–481.

Mockus, J. (1989). *Bayesian approach to global optimization: theory and applications*. Kluwer Academic Publishers.

Mockus, J., Tiesis, V., & Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards Global Optimisation*, *2*, 117–129.

Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed Gibbs: a linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research (JAIR)*, *64*, 705–748.

Ottens, B., Dimitrakakis, C., & Faltings, B. (2017). DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, *8*(5).

Petcu, A., & Faltings, B. (2005). DPOP: a scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 266–271.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

Rogers, A., Farinelli, A., Stranders, R., & Jennings, N. R. (2011). Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, *175*(2), 730–759.

Rust, P., Picard, G., & Ramparany, F. (2019). pyDCOP, a DCOP library for IoT and dynamic systems. In *International workshop on Optimisation in Multi-Agent Systems (OptMAS)*, Montréal, Canada.

Sarker, A., Arif, A. B., Choudhury, M., & Khan, M. M. (2020). C-CoCoA: a continuous cooperative constraint approximation algorithm to solve functional DCOPs. *arXiv*.

Sato, D. M., Borges, A. P., Márton, P., & Scalabrin, E. E. (2015). I-DCOP: train classification based on an iterative process using distributed constraint optimization. *Procedia Computer Science*, *51*, 2297–2306.

Sherman, J., & Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, pp. 124–127.

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 601–608.

Sultanik, E. A., Modi, P. J., & Regli, W. W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 247–253.

Törn, A., & Žilinskas, A. (1989). *Global optimization*, Vol. 350. Springer.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London.

van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In *Adaptation, Learning, and Optimization*, Vol. 12, pp. 207–251.

Van Leeuwen, C. J. (2017). CoCoA: a non-iterative approach to a local search (A)DCOP solver. In *Association for the Advancement of Artificial Intelligence*, pp. 3944–3950.

Vazquez, E., & Bect, J. (2010). Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and Inference (SPI)*, *140*(11), 3088–3095.

Vert, J.-P., Tsuda, K., & Schölkopf, B. (2004). A primer on kernel methods. *Kernel Methods in Computational Biology*, *47*, 35–70.

Vianna, L. G. R., Sanner, S., & de Barros, L. N. (2014). Continuous real time dynamic programming for discrete and continuous state MDPs. In *Brazilian Conference on Intelligent Systems (BRACIS)*, Vol. 3, pp. 134–139. IEEE.

Vinyals, M., Rodríguez-Aguilar, J. A., & Cerquides, J. (2009). Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Vol. 1, pp. 1239–1240.

Voice, T., Stranders, R., Rogers, A., & Jennings, N. R. (2010). A hybrid continuous max-sum algorithm for decentralised coordination. *Frontiers in Artificial Intelligence and Applications*, *215*, 61–66.

Wittenburg, L., & Zhang, W. (2003). Distributed breakout algorithm for distributed constraint optimization problems – DBArelax. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, p. 1158.

Yeoh, W., Feiner, A., & Koenig, S. (2010). BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research (JAIR)*, *38*, 85–133.

Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*, 53–65.

Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: formalization and algorithms. *Transactions on Knowledge and Data Engineering*, *10*(5), 673–685.

Zivan, R., Parash, T., & Naveh, Y. (2015). Applying max-sum to asymmetric distributed constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 432–439.