DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems

Brammert Ottens and **Christos Dimitrakakis** and **Boi Faltings**
EPFL, Lausanne, Switzerland
{first-name.last-name@epfl.ch}

Abstract

The Upper Confidence Bounds (UCB) algorithm is a well-known near-optimal strategy for the stochastic multi-armed bandit problem. Its extensions to trees, such as the Upper Confidence Tree (UCT) algorithm, have resulted in good solutions to the problem of Go. This paper introduces DUCT, a distributed algorithm inspired by UCT, for solving Distributed Constraint Optimization Problems (DCOP). Bounds on the solution quality are provided, and experiments show that, compared to existing DCOP approaches, DUCT is able to solve very large problems much more efficiently, or to find significantly higher quality solutions.

1 Introduction

The field of Distributed Constraint Optimization (DCOP) was introduced in (Yokoo et al. 1998) in its satisfaction form. A DCOP can be used to model distributed coordination problems. For example, in (Maheswaran, Pearce, and Tambe 2004) meeting scheduling problems are modeled as a DCOP. In (Ali, Koenig, and Tambe 2005) a sensor network problem is transformed into a DCOP, and in both (Ottens and Faltings 2008) and (Léauté, Ottens, and Faltings 2010) a transportation problem is solved using DCOP methods. Existing DCOP algorithms can be roughly subdivided into search based(SynchBB (Hirayama and Yokoo 1997), ADOPT (Modi et al. 2005)), inference based (DPOP (Petcu and Faltings 2005), O-DPOP (Petcu and Faltings 2006)) and local search based methods (MGM (Maheswaran, Pearce, and Tambe 2004), MGM2 (Maheswaran, Pearce, and Tambe 2004) and DSA (Zhang et al. 2005)).

This paper proposes a novel distributed approach for solving DCOPs, based on confidence bounds. A natural choice is the UCB (Auer, Cesa-Bianchi, and Fischer 2002) algorithm and its variants for tree-structured problems, such as UCT (Kocsis and Szepesvári 2006) and HOO (Bubeck et al. 2011). Such algorithms have been shown to be very successful in playing Go (Gelly and Silver 2008). However, they assume stochasticity or smoothness, which is not the case in our setting. We nonetheless show that UCB based search on DCOP not only significantly outperforms local search in terms of solution quality, but is also able to provide good feasible solutions for problems where optimal methods (in particular, DPOP) are too complex. We also provide a theoretical analysis of the proposed algorithms, based on weak assumptions on the problem structure.

To the best of our knowledge, no previous work uses sampling and confidence bounds for solving DCOPs. However, sampling has been used for solving both Constraint Satisfaction Problems (CSP) (Gogate and Dechter 2006) and Quantified CSPs (Baba et al. 2011). Furthermore, in (Léauté and Faltings 2011) sampling is used as a preprocessing method when dealing with stochastic problems.

The rest of this paper is organized as follows. Section 2 introduces the DCOP framework. Section 3 explains confidence bounds and describes the DUCT algorithm. Complexity results are given in Section 4, the experimental evaluation can be found in Section 5 and Section 6 concludes.

2 Distributed Constraint Optimization

A Distributed Constraint Optimization Problem (DCOP) consists of a set of variables, owned by different agents, and a set of constraints, each defined over a set of variables. The objective is to find a variable assignment that gives a feasible solution that minimizes cost. More precisely:

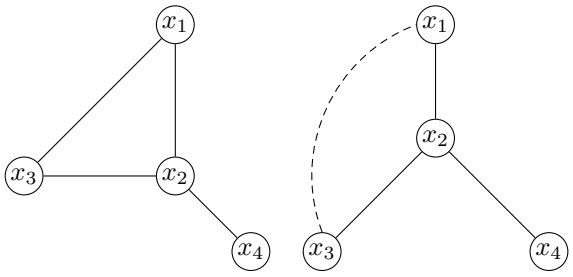
Definition 1. *A distributed constraint optimization problem (DCOP) is a tuple $\langle \mathcal{X}, \mathscr{D}, \mathcal{F} \rangle$ where*

- $\mathcal{X} \triangleq \{x_i \mid i = 1, \ldots, n\}$*, is a set of variables, to which the agents assign values.*
- $\mathscr{D} \triangleq \{D_i \mid i = 1, \ldots, n\}$*, is a collection of finite domains, with product space $\mathcal{D} = \prod_{i=1}^{n} D_i$, such that $x_i \in D_i$. Let $\mathbf{x} = (x_1, \ldots, x_n)$, with $\mathbf{x} \in \mathcal{D}$.*
- $\mathcal{F} \triangleq \{f_i \mid i = 1, \ldots, m\}$ *is a set of constraints. Each constraint $f_i : \mathcal{D}_i \to \mathbb{R} \cup \{\infty\}$ depends on $n(i)$ variables, with $\mathcal{D}_i \triangleq D_{i_1} \times \ldots \times D_{i_{n(i)}}$. We use $\mathbf{x}_{\|\mathcal{D}_i}$ for the projection of $\mathbf{x}$ to the subspace on which the i-th constraint is defined, and $\mathcal{X}_{\|i}$ for the variables in the range of f_i.*

Each variable is owned by a single agent, who selects its value. The objective is to find $\mathbf{x}$ minimizing the total cost:

$$f(\mathbf{x}) \triangleq \sum_{i=1}^{m} f_i\left(\mathbf{x}_{\|\mathcal{D}_i}\right), \tag{1}$$

with the minimum value being $f^*(\mathcal{D}) \triangleq \min_{\mathbf{x} \in \mathcal{D}} f(\mathbf{x})$.



(a) Constraint graph (b) pseudo-tree

Figure 1: The constraint graph for $f_1(x_1, x_2, x_3) + f_2(x_2, x_4)$ and one of its possible pseudo-trees

For ease of exposition, we assume that each agent owns only a single variable. However, our method can easily be extended to the more general case.

3 Distributed UCT

Notation and Concepts

The constraint structure can be captured using a constraint graph, defined below. An example can be seen in Fig. 1(a).

Definition 2 (Constraint Graph)**.** *Given a DCOP $\langle \mathcal{X}, \mathscr{D}, \mathcal{F} \rangle$, its constraint graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ is such that $(x_i, x_j) \in \mathcal{E}$ if there is a $f_k \in \mathcal{F}$ such that $x_i, x_j \in \mathcal{X}_{\|k}$*

In a DCOP, the global function f is decomposable in a set of factors, i.e. $f \triangleq f_1 + \ldots + f_m$. As f_i and f_j might depend on a common variable, a partial order on the variables could make the optimization more efficient. This can be obtained from the constraint graph by finding a *pseudo-tree* (Freuder and Quinn 1985) of the graph. A pseudo-tree $\mathcal{G}'$ is simply a rooted directed spanning tree on $\mathcal{G}$. In the algorithms we propose, agent communication takes place only via the edges in $\mathcal{G}'$. Any edge in $\mathcal{G} \setminus \mathcal{G}'$ is called a *back edge*. An example pseudo-tree is shown in Fig. 1(b), where the dashed line is a back edge. A factor is *enforced* by the lowest variable in the tree that influences it. For example, f_1 is enforced by x_3 while f_2 is enforced by x_4. The *separator* of a node n is the minimal set of nodes that must be removed to *separate* the subtree rooted at n from the rest of the tree. For example, the separator of x_3 is $\{x_2, x_1\}$.

The structure of the search space can be captured by AND/OR graphs (Dechter and Mateescu 2004). Such a graph consists of alternating AND nodes and OR nodes, where OR nodes represent alternative ways of solving the problem, and AND nodes represent a problem decomposition. The AND/OR graph of the pseudo-tree in Fig. 1 is shown in Fig. 2, with the squares and circles representing AND and OR nodes respectively. Note that since the subproblem rooted at x_4 does not depend on the value of x_1, we merge the two subgraphs corresponding to $x_1 = 0$ and $x_1 = 1$. A *path* represents an assignment to all variables. At AND nodes, the path *branches* to all the children, while at an OR node, the path *chooses* only a single child. The bold lines in Fig. 2 constitute a path, and represent the assignment $\{x_1 = 0\,, x_2 = 1\,, x_3 = 0\,, x_4 = 1\}$. To prevent confusion

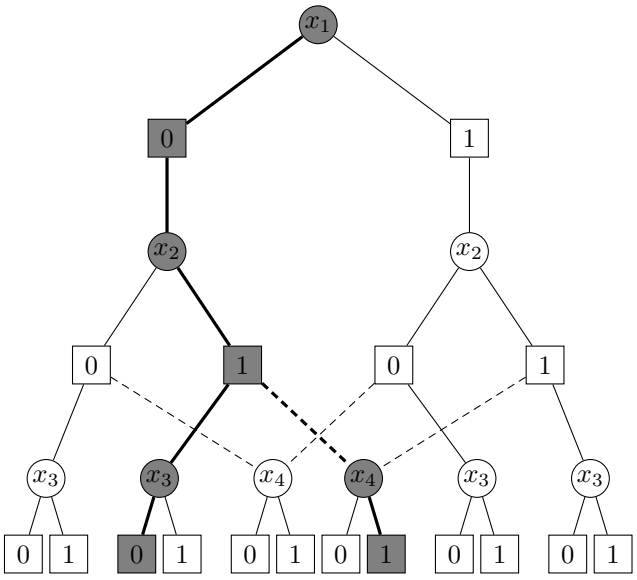


Figure 2: And AND/OR graph with $\forall_i D_i = \{0, 1\}$

between the pseudo-tree and the AND/OR graph, nodes in the pseudo-tree are from here on called agents, while nodes in the AND/OR graph are nodes.

Random Sampling: The RANDOM algorithm

DCOP search algorithms, such as ADOPT, operate by sending *context* messages down the pseudo-tree. A context is an assignment to all the variables in the receiving variable's separator. Based on this, variables systematically choose a value for themselves, in order to explore the search space. A different, but simple, approach would be to randomly sample the search space. We call this algorithm RANDOM.

The root agent starts sampling by selecting a value for its variable, and sending a CONTEXT message containing this value to all its children. Every time an agent k receives a CONTEXT message, containing a context a, it randomly chooses a value $d \in D_k$ for x_k, appends this to a, and sends this enlarged context to its children using a CONTEXT message. This process stops when the leaf agents are reached. At this time, the algorithm has selected a path through the AND/OR tree. Based on the received context a, the leaf agents now calculate the minimal value the sum of the constraints they enforce can take:

$$y_k^t = \min_{d \in D_k} \ell(a, d) \tag{2}$$

where t denotes that this is the t-th sample taken by agent k, and $\ell(a, d)$ is the sum of the values of the constraints enforced by agent k. y_k^t is sent up the path using a COST message. Every subsequent agent calculates its own COST message using the following equation:

$$y_k^t = \ell(a, d) + \sum_{k' \in C(k)} y_{k'}^t \tag{3}$$

Here $C(k)$ denotes the children of agent k. The cost is thus the sum of the samples reported by its children plus the value of its local problem.

Each agent k stores, for each context a it has received, the best value for y_k it has seen so far. More formally, let a_k^t be the context received by agent k at time t, and x_k^t the value chosen by agent k at time t, then each agent stores the following:

- $\hat{\mu}_{a,d}^t$: the lowest cost found for value d under context a:

$$\hat{\mu}_{a,d}^t \triangleq \min\left\{y_k^l \,\middle|\, l \leq t : a_k^l = a, x_k^l = d\right\}, \quad (4)$$

- $\hat{\mu}_a^t$: the lowest cost found under context a, $\hat{\mu}_a^t \triangleq \min_d \hat{\mu}_{a,d}^t$ and the value with the lowest cost $\hat{d}_a \triangleq \arg\min_d \hat{\mu}_{a,d}^t$

Using this information the agents together reconstruct, upon termination the best path seen.

Termination

As a DCOP is distributed, a local termination condition is necessary. To this end, the following additional information must be stored by each agent:

- $\tau_{a,d}^t$: the number of times value d has been selected for variable x_k under context a:

$$\tau_{a,d}^t \triangleq \sum_{l=1}^{t} \mathbb{I}\left\{a_k^l = a \wedge x_k^l = d\right\} \quad (5)$$

An agent k now terminates when the following two conditions are met:

- Its parent has terminated (this condition trivially holds for the root agent);
- The following equation holds for the last context a reported by its parent:

$$\max_{d \in D_k} \hat{\mu}_a^t - (\hat{\mu}_{a,d}^t - \sqrt{\frac{\ln \frac{2}{\delta}}{\tau_{a,d}^t}}) \leq \epsilon \quad (6)$$

  where ϵ and δ are parameters of the algorithm. Equation (6) holds when the biggest gap between the currently best value and the minimal lower bound on all values is smaller then ϵ. Note that, although the square root term has the form of a Hoeffding bound, it is used here heuristically.

When an agent k terminates, it adds $x_k = \hat{d}_a$ to the context a, and sends a F-CONTEXT message to its children, to signal its termination. Note that, since $\tau_{a,d}^t$ always increases, the algorithm is guaranteed to satisfy this condition in a finite number of steps.

Confidence Bounds: The DUCT algorithm

When an agent receives a context, one can say the search algorithm has reached an OR node in the AND/OR graph. In RANDOM, the next branch (variable value) on the path to the leaf nodes is chosen randomly. However, we could instead focus the search on more promising choices. One way to do that is by constructing a confidence bound B such that the best value for any context is at least B, and sampling the choice with the least bound. Indeed, the problem each OR node faces is similar to a *multi-armed bandit problem*, for which confidence bound based algorithms such as UCB (Auer, Cesa-Bianchi, and Fischer 2002) are nearly optimal solutions. UCB has been extended to tree-structured domains in the form of the UCB on trees (UCT) algorithm (Kocsis and Szepesvári 2006) and its variants BAST (Coquelin and Munos 2007) and HOO (Bubeck et al. 2011). These employ inequalities on the probability of large deviations of a sum of random variables from its expected value, which are used to optimistically search the space. The main assumptions are that the space is metric, the cost function is stochastic and its expected value satisfies a Lipschitz smoothness condition with respect to the metric. Our setting is quite dissimilar, mainly because it is hard to justify any smoothness assumption on typical DCOPs. However, under a different set of assumptions, described in Sec. 4, we construct a similar type of confidence bound, which results in an efficient distributed tree search algorithm (DUCT).

In order to use more informed bounds to guide the search, each agent k stores the following data for each context a:

- τ_a^t: the number of times context a has been received:

$$\tau_a^t \triangleq \sum_{l=1}^{t} \mathbb{I}\left\{a_k^l = a\right\} \quad (7)$$

We would like our bound to steer the sampling to the most promising values, while not completely ignoring the other values. Appropriate choices of this bound will result in a good balance between exploration of new branches and exploitation of currently known good branches. This can be achieved by a careful adjustment of the bound as t increases. Based on the collected information, we use the following bound, which is similar to the one employed in HOO (Bubeck et al. 2011)

$$B_{a,d}^t \triangleq \ell(a,d) + \max\{\hat{\mu}_{a,d}^t - L_{a,d}^t, \sum_{k' \in C(k)} B_{k'}^t\}, \quad (8)$$

where $B_{k'}^t = \min_{d' \in D_{k'}} B_{a',d'}^t$ is the bound reported by agent k' for context $a' = a \cup \{x_k = d\}$. For leaf agents, $B_{a,d}^t = \ell(a,d)$. Finally, the confidence interval $L_{a,d}^t$ is a UCB-style bound: (Auer, Cesa-Bianchi, and Fischer 2002)

$$L_{a,d}^t = \sqrt{\frac{2\lambda_a \ln \tau_a^t}{\tau_{a,d}^t}}, \quad (9)$$

where λ_a denotes the length of the path to the deepest leaf node. Unlike UCB, (9) is not due to statistical considerations, but due to a different assumption about the problem structure, given in the analysis of Sec. 4. This results in the bound (13), which has the same form as (9).

Bound (8) is an optimistic estimate of the optimal value for the sub-tree rooted at agent k for context a and choice d. Note that, when $B_{a,d}^t = \ell(a,d) + \hat{\mu}_{a,d}^t$, the optimal cost for the subtree rooted at agent k, for context a and choice d, has been found. After this occurs, d should be ignored under context a. More precisely, let $S_a^t = \{d \in D_k | B_{a,d}^t \neq \ell(a,d) + \hat{\mu}_{a,d}^t\}$ be the set of allowed sample values. After trying each value at least once (since otherwise (9) is undefined), the agent samples according to:

$$x_k^t \triangleq \arg\min_{d \in S_a^t} B_{a,d}^t, \quad (10)$$

Algorithm 1: DUCT algorithm for variable k

initialization
  if *root* **then**
    parentFinished = true;
    d = `sample()`;
    for *each child* k' **do**
      `send(`k'`, CONTEXT(`$\{x_k = d\}$`))`
  else
    parentFinished = false;
when `received(CONTEXT(`a`))` from parent
  if *variable* k *is a leaf node* **then**
    $\ell_{\min} = \min_{d \in D_k} \ell(a, d)$;
    `send(`*parent*`, COST(`ℓ_{min}, ℓ_{min}`))`
  else
    d = `sample(`a`)`;
    for *each child* k' **do**
      `send(`k'`, CONTEXT(`$a \cup \{x_k = d\}$`))`
when `received(F-CONTEXT(`a`))` from parent
  parentFinished = true;
  if *Eq.* (11) *satisfied* **then**
    for *each child* k' **do**
      `send(`k'`, F-CONTEXT(`$i \cup \{x_k = \hat{d}_a\}$`))`
  else
    d = `sample(`a`)`;
    for *each child* k' **do**
      `send(`k'`, CONTEXT(`$a \cup \{x_k = d\}$`))`
when `received(COST(`$y^t_{k'}$, $B^t_{k'}$`))` from child k'
  if *received cost message from all children* **then**
    if *parentFinished* $\wedge$ *Eq.* (11) *satisfied* **then**
      for *each child* k' **do**
        `send(`k'`, F-CONTEXT(`$i \cup \{\hat{d}_i\}$`))`
    else if *parentFinished* $\vee$ $y^t_k = \infty$ **then**
      d = `sample(`a`)`;
      for *each child* k' **do**
        `send(`k'`, CONTEXT(`$a \cup \{x_k = d\}$`))`
    else
      `send(`*parent*`, COST(`y^t_k, $B^t_{k,}$`))`

with ties broken randomly. Also, the termination condition is now:

$$\max_{d \in S^t_a} \hat{\mu}^t_a - (\hat{\mu}^t_{a,d} - \sqrt{\frac{\ln \frac{2}{\delta}}{\tau^t_{a,d}}}) \leq \epsilon. \quad (11)$$

The description of DUCT can be found in Algorithm 1. $parentFinished$ is used to store whether the agent's parent has terminated.

Normalization

The above assumes that the range of the *global* cost is $[0, 1]$. As general DCOP problems, this is not the case, a normalization procedure must be carried out beforehand. An upper bound on the global cost can be found by summing up the maximal values of all the local cost functions. This can be done in a bottom-up procedure: As soon as the root of the pseudo-tree has received this upper bound, UB, every local constraint is shifted to the left by subtracting its minimal value, and then divided by UB. During this phase, we can also obtain auxiliary information, such as the depth of the deepest descendant.

Hard Constraints

Hard constraints, where there is some infeasible set $\mathcal{D}_I \subset \mathcal{D}$ such that $\forall \mathbf{x} \in \mathcal{D}_I \; f(\mathbf{x}) = \infty$ break the normalization procedure. One way to deal with this is to adjust the cost function such that all costs remain finite, while ensuring that $f(\mathbf{x}) < f(\mathbf{x}')$ for every $\mathbf{x} \notin \mathcal{D}_I$, $\mathbf{x}' \in \mathcal{D}_I$, i.e. feasible assignments have lower cost than infeasible assignments. However, normalization will squeeze the set of feasible costs into a small range, making optimization harder. As we verified experimentally, replacing infeasible costs with a penalty resulted in the algorithm minimizing constraint violations instead of looking for good feasible solutions.

For this reason, firstly we only normalize so that $f(x) \in [0, 1]$ for $x \notin \mathcal{D}_I$. Secondly, all infeasible parts of $\mathcal{D}$ are *pruned* as soon as their infeasibility is evident, and a search for the next feasible solution is started. Infeasible assignments can easily be recognized by the variable that enforces the infeasible constraint. Thirdly, when a node a does not have an infeasible local problem, but all its children reported to be infeasible, a is considered to be infeasible as well.

4 Theoretical Analysis

A general problem-independent analysis for DCOP is not possible. One can always construct a counterexample such that there exists only one optimal solution $\mathbf{x}^*$, with all remaining choices of $\mathbf{x}$ being either infeasible or far from optimal. For that reason, we assume that the number of suboptimal choices are bounded as follows:

Assumption 1. *Let λ be the counting measure on $\mathcal{D}$ and let $f^*(A) \triangleq \min\{f(\mathbf{x}) \,|\, \mathbf{x} \in A\}$, for all $A \subset \mathcal{D}$. There exists $\beta > 0$ and $\gamma \in [0, 1]$ s.t. $\forall A \subset \mathcal{D}$, $\epsilon \geq \gamma^{1/\beta}$:*

$$\lambda(\{\mathbf{x} \in A \,|\, f(\mathbf{x}) > f^*(A) + \epsilon\}) \leq \lambda(A)\, \gamma \epsilon^{-\beta}. \quad (12)$$

This assumption is quite weak, as it does not guarantee how many choices will be arbitrarily close to the optimal. It is weaker than a Lipschitz smoothness condition, as it is does not require a metric. Intuitively, it allows the optimal solution to be hidden in a bad set of solutions, as long as there are sufficiently many good solutions elsewhere. The assumption even holds in pathological cases, by taking β, γ close to $0, 1$ respectively. Nevertheless, it can be used to obtain bounds on the value $f^*(A)$ of any set A. Assume that we have taken T samples from A, with values $(y^1_k, \ldots, y^T_k)$. In the worst-case, these are the T worst values, and thus an upper bound on the gap between the best-found value and the optimal value in A is:

$$\epsilon \leq \left(\frac{\gamma \lambda(A)}{T}\right)^{1/\beta}. \quad (13)$$

Definition 3. *Let $\rho(T)$ be the regret after T steps:*

$$\rho(T) \triangleq \min\left\{y_0^t \,\middle|\, t = 1, \ldots, T\right\} - f^*(\mathcal{D}). \quad (14)$$

Thus, an algorithm that samples all of $\mathcal{D}$ in fixed arbitrary order has regret bounded by (13) with $A = \mathcal{D}$, since an adversary could manipulate the problem so that the worst values are seen first. On the contrary, the regret of RANDOM, which selects assignments uniformly, i.e.: $\mathbf{x}_t \sim \mathrm{Uni}(\mathcal{D})$, does not depend on the size of $\mathcal{D}$:

Theorem 1. *The expected regret of the* RANDOM *algorithm is* $\mathbb{E}\,\rho(T) \in O(1/\beta T + \gamma^{1/(\beta+1)})$.

Proof. If Ass. 1 holds, then the probability that the regret of RANDOM after T samples exceeds ϵ is bounded by:

$$\mathbb{P}\left(\bigwedge_{t=1}^{T} f(\mathbf{x}_t) > f^*(\mathcal{D}) + \epsilon\right) \leq \left(\gamma\epsilon^{-\beta}\right)^T. \quad (15)$$

It follows from (15) and the fact that the regret and probabilities are bounded that $\forall\epsilon$, $\mathbb{E}\,\rho(T) \leq \left(\gamma\epsilon^{-\beta}\right)^T + \epsilon$. Setting the derivative to zero, we find that $\epsilon_0 = \left(\beta T\gamma^T\right)^{1/(\beta T+1)}$. Replacing to find the tightest bound:

$$\mathbb{E}\,\rho(T) \leq \gamma^T(\beta T\gamma^T)^{-\frac{\beta T}{\beta T+1}} + (\beta T\gamma^T)^{\frac{1}{\beta T+1}} \quad (16)$$

$$\leq \frac{1}{\beta T} + (\beta T\gamma^T)^{\frac{1}{\beta T+1}} \quad (17)$$

The second term can be bounded as follows. Note that $\max_x x^{1/(1+x)} < \max_x x^{1/x}$ and that

$$\frac{d}{dx}x^{1/x} = \frac{d}{dx}e^{\frac{1}{x}\ln x} = e^{\frac{1}{x}\ln x}x^{-2}(1 - \ln x),$$

which has a root $x = e$. Consequently, $(\beta T)^{1/(\beta T+1)} < e^{1/e}$. Finally, $\gamma^{T/(\beta T+1)} \leq \gamma^{1/(\beta+1)}$. □

Let us now consider DUCT. Since we have no knowledge of β, γ, the algorithm increases (9) slowly, so that at some point it will bound (13). For that reason, DUCT is initially optimistic and searches mostly subsets where it has already found good solutions, because it believes its bounds are tight. The following theorem makes this intuition precise, by bounding the time spent in suboptimal parts of the space:

Theorem 2. *Consider two disjoint sets $A_1, A_2 \subset \mathcal{D}$ with $f^*(A_1) = f^*(A_2) + \Delta$, $A \triangleq A_1 \cup A_2$. If τ is the number of times A is sampled then the number of times the suboptimal set A_1 is sampled under* DUCT *is $O(\Delta^{-2}\ln\tau + (\gamma/2)^{\beta/(2-\beta)} + e^{\gamma/2})$.*

Proof. Let $\hat{\mu}_i^t$ be the best-measured value in A_i at time t. For our bounds to hold, we must sample A at least $e^{\gamma/2}$ times and A_i at least $\tau_i = (\gamma/2)^{\beta/(2-\beta)}$ times. Then, we have:

$$f^*(A_i) \in [\hat{\mu}_i^t - L_{i,t}, \hat{\mu}_i^t + L_{i,t}],$$

Since we select A_1 rather than A_2, we have: $f^*(A_2) \geq \hat{\mu}_2^t - L_{2,t} \geq \hat{\mu}_1^t - L_{1,t}$. But $\hat{\mu}_1^t \geq f^*(A_1) - L_{1,t}$. Consequently, in order to select A_1 it is necessary that $f^*(A_2) \geq f^*(A_1) - 2L_{1,t}$. From the definition, we obtain $\tau_i \leq \left(\frac{4}{\Delta}\right)^2 \ln\tau$. □

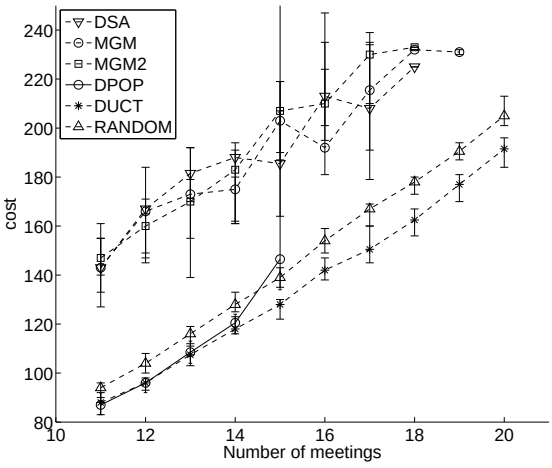


Figure 3: Meeting scheduling: Solution cost

To complete the analysis, we prove a lower bound on the expected regret of any algorithm.

Theorem 3. $\mathbb{E}\,\rho(T) \in \Omega(\gamma^{1/2\beta+T/2})$.

Proof. It is sufficient to consider a function g in only one variable, for which Ass. 1 holds with equality. Assume a random bijection $h : \mathcal{D} \to \mathcal{D}$. Then function $f(\mathbf{x}) = g(h(\mathbf{x}))$ also satisfies Ass. 1 and any algorithm is equivalent to RANDOM. Then $\mathbb{E}\,\rho(T) \geq \gamma^T\epsilon^{1-\beta T}$ as (12) holds with equality. Selecting $\epsilon = \gamma^{1/2\beta}$ completes the proof. □

nodes	18	19	20	21	22	23	24	25	26
DPOP	0	0	0	0	0	-	-	-	-
DUCT	0	0	0	0	0	0	0	0	0
RANDOM	0	0	0	1	1	1	2	2	2
DSA	2	2	2	2	2	2	2	2	2
MGM	2	2	2	2	2	2	2	2	2
MGM2	1	0	0	0	0	0	0	0	0

Table 1: Graph coloring: Number of constraint violations

5 Experimental Evaluation

The introduced algorithms are evaluated on the *meeting scheduling* domain introduced in (Maheswaran et al. 2004), and on randomly generated *graph coloring* problems. DUCT is compared against both optimal algorithms (DPOP, O-DPOP, ADOPT, SynchBB and AFB) and non-optimal algorithms (DSA, MGM and MGM2). ADOPT, SynchBB and AFB could not solve the majority of the problems within the given time, and are thus omitted from the figures. O-DPOP found the same solutions as DPOP, and so is only included in the CPU time graphs. Furthermore, RANDOM is used as a baseline sampling algorithm. Comparisons are made both on solution quality and runtime, where runtime is measured using the *simulated time* notion as introduced in (Sultanik, Lass, and Regli 2007).

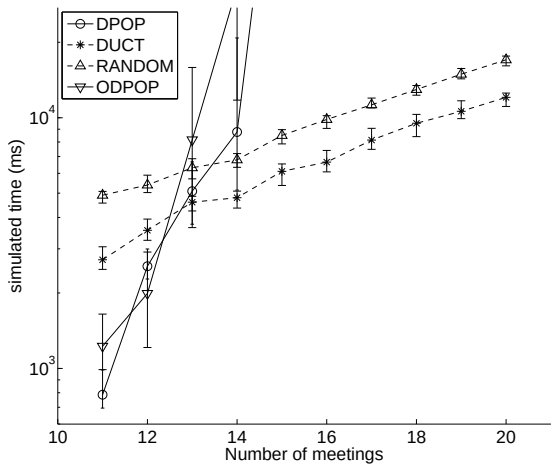


Figure 4: Meeting scheduling: Simulated time in ms

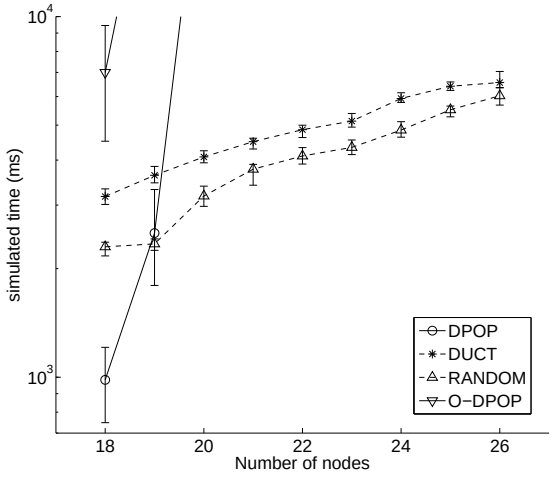


Figure 5: Graph coloring: Simulated time

Experimental Setup

All evaluated algorithms are implemented in the FRODO platform (Léauté, Ottens, and Szymanek 2009). For *meeting scheduling*, we used a pool of 30 agents, with 3 agents per meeting and between 11 and 20 meetings per instance. The cost for each meeting is randomly taken from $[0, 10]$. For the *graph coloring* problems, graphs with a density of 0.3 were randomly generated, the number of available colors was 5 and between 18 and 26 nodes per instance. Problems were modelled as Min-CSPs. For each set of parameters, 99 problem instances were generated using the generators provided by FRODO, with a time out set at 15 minutes per problem. The parameters δ and ϵ were set to 0.05. The experiments are run on a 64 core Linux machine, with 2GB per run.

DSA, MGM and MGM2 do not have any termination condition, and, to be fair, were given as much time as DUCT.

The median is reported, together with 95% confidence intervals. In some instances, the optimal algorithms time out and consequently, their median can be higher than DUCT.

Experimental Results

Figure 3 shows the performance on *meeting scheduling*. It can be seen that DPOP is only able to solve problems up to 14 meetings, after which it times out, while O-DPOP is even slower. One can also see that DUCT finds optimal to near optimal solutions for those problems. In fact, the heuristically used stopping criterion (6) appears to hold in practice, as 93% of those runs fall within the 5% error range set as the target for DUCT. This means the assumptions underlying the bound used characterize the search space well. Overall, DUCT consistently performs better than RANDOM, which in turn outperforms the local search algorithms[1]. Figure 4 shows the amount of simulated time used by each algorithm (since the local search algorithms are given the same time as DUCT, they are excluded). It is clear that, for this domain, DUCT terminates earlier than RANDOM, while the optimal algorithms have an advantage for small problems.

Table 1 and Fig. 5 show the cost and time respectively for the *graph coloring* domain. In this case, DUCT always finds the optimal solution, while RANDOM's performance degrades as the size of the problem increases. DUCT stops later in this problem than RANDOM, probably due to focusing on suboptimal branches in the initial stages. The optimal algorithms again fail to scale to larger problems. While the other local algorithms fail, MGM2 performs well on this domain. Since MGM2 is designed to find a 2-coordinated solution this is perhaps not surprising.

In general, DUCT finds nearly optimal solutions, but scales much better than optimal methods, both in terms of time and message size. Due to space considerations, message size results have been omitted, but it scales similarly to time. The performance of the local search algorithms was mostly poor, since they could only find feasible solutions for the easiest problems, in contrast to the sampling algorithms.

6 Conclusions

We introduced a confidence bound based approach for solving Distributed Constraint Optimization Problems, inspired by Monte-Carlo tree search and the application of UCB to tree-structured problems. The result, a Distributed UCT algorithm (DUCT), takes advantage of the distributed and deterministic nature of the problem, as well as the hard constraints, in order to efficiently search the solution space.

Theoretically, we show that even though DCOP is not a smooth domain, we can still obtain meaningful bounds on the regret, by using an assumption on the number of bad solutions. However, the gap between the upper and lower bounds indicates that even better results may be possible. It is our view that a more intricate analysis requires additional assumptions, such as some notion of smoothness, which we

[1]Local search algorithms failed to find a feasible solution in more than 50% of the instances. Hence only the solved instances are displayed in Fig 3

shall investigate in further work.

The experiments show that DUCT not only can obtain good solutions within a reasonable amount of time, but also that it performs significantly better than local search within the same time constraints. They also show that DUCT can handle much bigger problems than optimal algorithms, and performs nearly as well for smaller domains, making it a very suitable algorithm for practical use. In all, DUCT is thus very well suited for solving DCOPs.

Acknowledgements This work was funded by the Swiss National Science Foundation under Project Number 200020-129515 and the EU FP7 Marie-Curie IEF project ESDEMUU, Grant Number 237816.

References

Ali, S.; Koenig, S.; and Tambe, M. 2005. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS '05*, 1041–1048.

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

Baba, S.; Joe, Y.; Iwasaki, A.; and Yokoo, M. 2011. Real-time solving of quantified CSPs based on monte-carlo game tree search. In *IJCAI*, 655–661.

Bubeck, S.; Munos, R.; Stoltz, G.; and Szepesvári, C. 2011. X-armed bandits. *Journal of Machine Learning Research* 12:1655–1695.

Coquelin, P.-A., and Munos, R. 2007. Bandit algorithms for tree search. In *UAI '07, Proceedings of the 23rd Conference in Uncertainty in Artificial Intelligence, Vancouver, BC Canada*.

Dechter, R., and Mateescu, R. 2004. Mixtures of deterministic-probabilistic networks and their and/or search space. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, 120–129. Arlington, Virginia, United States: AUAI Press.

Freuder, E. C., and Quinn, M. J. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*, 1076–1078. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9×9 computer go. In *Proceedings of the 23rd National Conference on Artificial intelligence - Volume 3*, 1537–1540. AAAI Press.

Gogate, V., and Dechter, R. 2006. A new algorithm for sampling CSP solutions uniformly at random. In Benhamou, F., ed., *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 711–715.

Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In *CP'97*, 222–236.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning, (ECML06)*.

Léauté, T., and Faltings, B. 2011. Distributed constraint optimization under stochastic uncertainty. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI'11)*, 68–73.

Léauté, T.; Ottens, B.; and Faltings, B. 2010. Ensuring privacy through distributed computation in multiple-depot vehicle routing problems. In *Proceedings of the ECAI'10 Workshop on Artificial Intelligence and Logistics (AILog'10)*.

Léauté, T.; Ottens, B.; and Szymanek, R. 2009. FRODO 2.0: An open-source framework for distributed constraint optimization. In Hirayama, K.; Yeoh, W.; and Zivan, R., eds., *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, 160–164.

Maheswaran, R. T.; Tambe, M.; Bowring, E.; Pearce, J. P.; and Varakantham, P. 2004. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, 310–317.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2004. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of ISCA PDCS'04*, 432–439.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161:149–180.

Ottens, B., and Faltings, B. 2008. Coordinating agent plans through distributed constraint optimization. In *Proceedings of the ICAPS'08 Multiagent Planning Workshop (MASPLAN'08)*.

Petcu, A., and Faltings, B. 2005. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 266–271.

Petcu, A., and Faltings, B. 2006. O-DPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, 703–708.

Sultanik, E. A.; Lass, R. N.; and Regli, W. C. 2007. DCOPolis: A framework for simulating and deploying distributed constraint optimization algorithms. In Pearce, J. P., ed., *Proceedings of the 9th Intl Workshop on Distributed Constraint Reasoning (CP-DCR'07)*.

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

Zhang, W.; Wang, G.; Xing, Z.; and Wittenburg, L. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161(1–2):55–87.