Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

CoCoA: A Non-Iterative Approach to a Local Search (A)DCOP Solver

Cornelis Jan van Leeuwen
TNO and Delft University of Technology
Eemsgolaan 3, Groningen, The Netherlands
coen.vanleeuwen@tno.nl

Przemysław Pawełczak
Delft University of Technology
Mekelweg 4, Delft, The Netherlands
p.pawelczak@tudelft.nl

Abstract

We propose a novel incomplete cooperative algorithm for distributed constraint optimization problems (DCOPs) denoted as Cooperative Constraint Approximation (CoCoA). The key strategy of the algorithm is to use a semi-greedy approach in which knowledge is distributed amongst neighboring agents, and assigning a value only once instead of an iterative approach. Furthermore, CoCoA uses a unique-first approach to improve the solution quality. It is designed such that it can solve DCOPs as well as Asymmetric DCOPS, with only few messages being communicated between neighboring agents. Experimentally, through evaluating graph coloring problems, randomized (A)DCOPs, and a sensor network communication problem, we show that CoCoA is able to very quickly find solutions of high quality with a smaller communication overhead than state-of-the-art DCOP solvers such as DSA, MGM-2, ACLS, MCS-MGM and Max-Sum. In our asymmetric use case problem of a sensor network, we show that CoCoA not only finds the best solution, but also finds this solution faster than any other algorithm.

Introduction

Distributed Constraint Optimization Problems (DCOPs) is a class of optimization problems in which discrete variables are controlled by distributed agents and the optimization function itself operates over the complete set of variables (Hirayama and Yokoo 1997). DCOPs are encountered in many fields such as in wireless LAN channel allocation (Yeoh and Yokoo 2012), coordination of mobile sensing teams (Yedidsion, Zivan, and Farinelli 2014) or coordination of tasks (Farinelli et al. 2008). By definition of DCOP, the involved agents are part of a team and need to cooperate in order to perform well on the global task. Usually in DCOPs, cooperation between agents is achieved by passing messages from one agent to another.

A number of complete algorithms have been proposed to find the optimal solution of a DCOP, amongst which are ADOPT (Modi et al. 2005), DPOP (Petcu and Faltings 2005), NCBB (Chechetka and Sycara 2006) and Asynchronous Forward Bounding (Gershman, Meisels, and Zivan 2009). However, DCOP problems are NP-hard (Modi 2003), so the effort to find the optimal solution becomes intractable for increasingly large-scale problems. Therefore, incomplete DCOP algorithms trade a distance from the optimal solution for convergence speed and are thus more suited for large-scale problems. Examples of incomplete DCOP solvers are DBA (Yokoo et al. 1998), DSA (Fitzpatrick and Meertens 2003), Max-Sum (Farinelli et al. 2008), MGM and MGM-2 (Maheswaran, Pearce, and Tambe 2004).

Recently, an extension on the DCOP framework has been described in which agents may value a set of constrained assignments differently. In these Asymmetric DCOPs (ADCOPs) constraints have different costs for their agents. The ACLS and MCS-MGM algorithms (Grubshtein et al. 2010; Grinshpoun et al. 2013) have been proposed to enable solving this class of problems. More recently it has also been shown that the Max-Sum_ADVP algorithm can solve ADCOPs (Zivan, Parash, and Naveh 2015).

In this paper, a motivating case study is to find an optimal configuration of a sensor network, tasked with monitoring the cargo of a shipping container for an extensive period of time. Since in sensor networks the communication between nodes is of the largest influence on the battery lifetime, we need to minimize the communication between nodes during the optimization process, and require an algorithm that is guaranteed to converge to a solution as quickly as possible. However, most existing DCOP algorithms use an iterative approach, which requires many rounds of message passing during the optimization process making them unsuitable for this case study.

In this paper we introduce a new DCOP algorithm, denoted as Cooperative Constraint Approximation (CoCoA), which uses a non-iterative, semi-greedy approach with a one-step look ahead. We show that it can not only cope with asymmetric constraints, but also finds high quality solutions much faster than other (A)DCOP solvers. Experimentally we show that in some cases this leads to a reduction of up to two orders of magnitude number of transmitted messages and cost function evaluations, thus leading to superior running times.

DCOP: Problem Statement and Challenges

DCOPs are defined as a tuple $\mathcal{T} = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ in which $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$ and $\mathcal{X}$ is a finite set of variables $\{X_1, X_2, \ldots, X_n\}$ with finite discrete domains $\{D_1, D_2, \ldots, D_n\}$ from $\mathcal{D}$ such that $X_i \in D_i$. Each agent

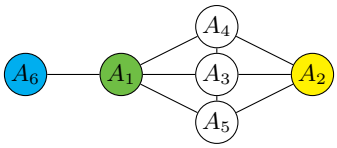

Figure 1: Example of a constraint graph in a graph coloring problem with six agents (vertices), variables (colors), and nine constraints (edges) between them.

A_i is assigned one variable X_i, and therefore $|\mathcal{A}| = |\mathcal{X}| = |\mathcal{D}|$. Then, $\mathcal{R}$ is a set of relations (constraints) in which each constraint $C \in \mathcal{R}$ defines a non-negative cost. For DCOPs such costs are defined for every possible value assignment of a set of variables $C\colon\ D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0}$, while for ADCOPs each constraint defines a set of costs for each involved variable, i.e. $C\colon\ D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^k_{\geq 0}$. Having all definitions of $\mathcal{T}$, in (A)DCOPs the goal of the agents is to minimize the global cost function, i.e.

$$\arg\min_{\mathcal{X}} \sum \mathcal{R}. \tag{1}$$

In the rest of this paper we shall only take into account binary constraints, in which exactly two variables are considered, of the form $C_{i,j}\colon\ D_i \times D_j \rightarrow \mathbb{R}^2_{\geq 0}$.

Definitions We refer to agents as *neighbors* if there is a constraint between their corresponding variables. This follows the real-life situation of limited range between agents, e.g. communication range in wireless networks. The set of all neighbors of an agent $\mathcal{M}_i \subseteq \mathcal{A}$ is called the *neighborhood*. The set $\hat{\mathcal{X}}_i$ denotes the set of known assigned values of the neighbors of A_i and is also referred to as the *current partial assignment* (CPA). Note that the constraints between variables can be shown as an undirected graph, see Figure 1.

DCOP: Existing Solvers

DCOP solvers can be categorized into *complete* and *incomplete*. Complete solvers search the entire solution space and are guaranteed to find the optimal solution, while incomplete solvers try to find a "good" solution in a reasonable time.

Incomplete solvers such as DSA (Fitzpatrick and Meertens 2003), MGM-2 (Maheswaran, Pearce, and Tambe 2004), ACLS or MCS-MGM (Grubshtein et al. 2010) are local search algorithms, trying to approximate the global function by solving a local problem. DSA is known for its low communication overhead and its ability to find high quality solutions for symmetric DCOPs (Pearce and Tambe 2007), whereas ACLS and MCS-MGM can also solve ADCOPs.

The Max-Sum algorithm (Farinelli et al. 2008) is another incomplete solver, that works in a completely different manner. It operates on a bipartite graph, separating variable from constraint nodes, and spreads information through the graph to estimate the effect of value assignments. It is capable of finding very high quality solutions, but only when the graphs contain no cycles. In order to deal with cyclic graphs and asymmetric costs, variations of the algorithm have been proposed such as Max-Sum_ADVP (Zivan and Peled 2012). Max-Sum and the local search algorithms apply an *iterative approach*; they evaluate their performance, share information, update their variable, and repeat until a stopping criterion is met—usually a predetermined number of iterations.

Challenges

In this paper we propose an algorithm that is capable of solving (A)DCOPs with a minimal communication and computation overhead. We hypothesize we can achieve this by *not* iteratively sending messages and updating the variable, but instead using a greedy, one-step look ahead approach. This means that each agent takes a decision based on information only from its direct neighbors and will activate agents sequentially. Under these circumstances we need to address the following challenges:

Challenge 1: Premature Assignment A non-iterative DCOP solver assigns a value to a variable only once. In its most simple form an agent would only look at its local constraint costs and the known values of its neighbors. It would select a value that minimizes its local cost and update its variable. Greedy algorithms have the advantage of converging very fast, but early choices may turn out to be suboptimal when neighbors have assigned their value.

Challenge 2: Synchronization When two neighbor agents are both deciding for a new value a race condition may occur, i.e. the outcome of one agent arrives too late for the other agent's decision. For iterative algorithms this is not an issue since in a later iteration one agent can correct for any incorrect assumptions; for non-iterative solver this is not possible.

Challenge 3: Asymmetric Costs An incomplete solver may end up in a local minimum if a local beneficial assignment leads to poor global results. An agent may assign a variable to decrease its local cost, but potentially increases the cost of its neighbors. This *over-greediness* may cause the global cost to increase or lead to unstable solutions. In strongly asymmetric constraints for every combination of value assignments at least one agent can improve its local cost by shifting to another assignment without decreasing the global cost. Under these circumstances iterative solvers may not converge to the minimum where both agents are assigned the same value, but agents on either side of the constraint will maintain a cycle of assigning different values to improve their local cost.

CoCoA: A New DCOP Solver

To address the challenges introduced in the previous section we propose a new incomplete ADCOP algorithm based on a semi-greedy strategy, denoted as *Cooperative Constraint Approximation* (CoCoA), employing three key ideas:

1. *A one-step look ahead* to consider the effect of an assignment on the cost of neighbors. This is especially effective when a neighbor is constrained in its choices;
2. *A unique-first* approach, such that an agent will only assign a value if it is a unique local optimum for its variable. If it cannot find a unique solution, the decision will be delayed until more information is available;
3. *A state machine* to spread and keep track of the algorithm's activity, prevent dead-locks or endless loops.

By minimizing not only the local cost, but also the cost of its single-hop neighbors, we hypothesize that CoCoA can find a solution with a low global cost with relatively little overhead. When CoCoA is triggered it first inquires its neighbors what the effect of different assignments would be for their local cost. The neighboring agents decide the resulting cost effect asynchronously and return to the inquirer what the minimum effect would be. Upon receiving the estimated costs the active agent will assign the value that minimizes the sum of all incurred costs, including its own.

During the variable assignment process it is possible that multiple values suit equally well, especially in an early stage of the algorithm when multiple neighbors have no assigned values. In such cases the assignment will be postponed until a neighbor has changed its value. We hypothesize that this *unique-first* approach will help in avoiding premature convergence to a sub-optimal solution. However, this approach may lead to a deadlock if all the neighbors are waiting for each other. Therefore we partition the algorithm in four *states* and have agents inform their neighbors about their internal state. When all neighbors are in the HOLD state, a bound denoting the "allowed uniqueness" is increased until a decision can be made. The proposed algorithm is given in pseudocode in Algorithm 1, with accompanying set of messages and agent states in Table 1 and Table 2, respectively and discussed in detail in the subsequent section.

Remark: Due to its non-iterative approach, it is impossible to recover from early choices. This may lead to situations in which a variable *must* be set to a value that leads to a very high cost, i.e. it has to break a hard constraint. Therefore we expect that CoCoA may not perform well in a problem with many hard constraints.

CoCoA: Algorithm Description

CoCoA assumes that all neighbors A_i can communicate with, $\mathcal{M}_i$, are known. Also, each $A_j \in \mathcal{M}_i$ must know its neighbor's domain D_i. Finally we assume that all nodes are reachable from any other node, i.e. for every pair of nodes there must be a set of edges that connects them.

Initially all agents start in the IDLE state and activation occurs at any random node. As soon as one agents finishes the algorithm it will trigger the algorithm for its neighbors. When any A_i has to find an assignment, it will first send an **INQMSG**$(i, \hat{\mathcal{X}}_i)$ message to its neighbors (line 3 of Algorithm 1). This will trigger agents $A_j \in \mathcal{M}_i$ to calculate for every possible assignment for X_i what the lowest cost would be for A_j taking into account the CPA and that assignment for X_i. That is, each $A_j \in \mathcal{M}_i$ calculates for every $X_{i,k} \in D_i$

$$\Theta_{j,k} = \min_{X_{j,l} \in D_j} \sum_{C \in \mathcal{R}_j} C\left(\hat{\mathcal{X}}_j \cap X_{i,k} \cap X_{j,l}\right), \qquad (2)$$

where $X_{i,k}$ denotes that X_i is assigned the kth value of D_i.

If variables are not yet assigned in $\hat{\mathcal{X}}_j$, the cost of their constraints can not be determined and the mean cost is used, i.e. a *one*-step look ahead is performed. The resulting cost map $\Theta_j = \{\Theta_{j,1}, \Theta_{j,2}, \ldots, \Theta_{j,|D_i|}\}$ is sent via a **COSTMSG**(Θ_j) back to the inquiring A_i. Then, A_i finds X_i by

$$\delta = \arg\min_k \sum_{j=1}^{|\mathcal{M}_i|} \Theta_{j,k}, \qquad (3)$$

and assigning $X_{i,\delta}$. The minimizing value may achieve a minimum for more than one value of X_i since the $\arg\min$ operator can return a set of minimizers. The *uniqueness* of this minimal cost is the number of distinct values that achieve this minimum, defined as $\mathcal{U}(X_i) = |\delta|$.

This uniqueness will be compared with a bound β to determine if this solution is accepted (line 6 of Algorithm 1). Initially $\beta = 1$, so that only unique solutions are accepted. If $\beta < \mathcal{U}(X_i)$ and at least one neighbor is ACTIVE or IDLE the algorithm switches to the HOLD state and waits until another node has updated its state to DONE before the algorithm is run again. If an **UPDSTATE**$(j, \texttt{HOLD})$ message is received, indicating that the last neighbor is in the HOLD state, then β is increased by one and the algorithm is repeated (line 21 of Algorithm 1). This mechanism makes sure that premature choices are avoided until more information is available. Initially, when no agents have a value assigned this may occur frequently, but as more variables are set the chances of such *impasses* decrease.

If $\mathcal{U}(X_i) \leq \beta$ then X_i is chosen randomly from all minimizers and is communicated to neighbors $A_j \in \mathcal{M}_i$ in a **SETVAL**(i, X_i) message. This makes the neighbors A_j update their CPA (as they now know the value of X_i) and triggers the algorithm for them.

Algorithm 1 CoCoA Algorithm

When started **or** upon receiving **UPDSTATE**$(j, \texttt{DONE})$ at on A_i:
Require: state:=IDLE **or** state:=HOLD
1: state$\leftarrow$ ACTIVE
2: send $\forall A_j \in \mathcal{M}_i$ **UPDSTATE**$(i, \texttt{ACTIVE})$
3: send $\forall A_j \in \mathcal{M}_i$ **INQMSG**$(i, \hat{\mathcal{X}}_i)$
4: wait for all **COSTMSG**(Θ_j)
5: find δ using (3)
6: **if** $\mathcal{U}(X_i) \leq \beta$ **or** number of idle/active neighbors is 0 **then**
7: $X_i \leftarrow$ random from $X_{i,\delta}$
8: state$\leftarrow$ DONE
9: send $\forall A_j \in \mathcal{M}_i$ **UPDSTATE**$(i, \texttt{DONE})$
10: send $\forall A_j \in \mathcal{M}_i$ **SETVAL**$(i, \hat{\mathcal{X}}_i)$
11: **else**
12: state$\leftarrow$ HOLD
13: send $\forall A_j \in \mathcal{M}_i$ **UPDSTATE**$(i, \texttt{HOLD})$
14: **end if**

Upon receiving **INQMSG**$(i, \hat{\mathcal{X}}_i)$ at A_j:
15: **for all** $X_{i,k} \in D_i$ **do**
16: find $\Theta_{j,k}$ using (2)
17: **end for**
18: Send A_i **COSTMSG**(Θ_j)

Upon receiving **UPDSTATE**(i, S):
19: Store state S of neighbor A_i
20: **if** S is HOLD **and** my state is HOLD **and** number of idle/active neighbors is 0 **then**
21: β++
22: Repeat algorithm
23: **else if** S is DONE **and** my state is HOLD **then**
24: Repeat algorithm
25: **end if**

Table 1: List of messages sent by CoCoA

Message	Description
INQMSG$(i, \hat{\mathcal{X}}_i)$	Sent by A_i at start of algorithm
COSTMSG(Θ_j)	Neighbors' reply; contains Θ_j
SETVAL(i, X_i)	Indicator: A_i assigned a value to X_i
UPDSTATE(i, S)*	Sent when A_i updates its state to S

Table 2: Agents' potential states in CoCoA

State	Description
IDLE	Agent's default/initial state; indicates agent is active, but not yet started.
ACTIVE	State after agent's activation; finding an assignment for its variable.
HOLD	At an impasse; delay variable assignment and await information from neighbors.
DONE	Final CoCoA state; indicates that agent has assigned a final value to its variable.

CoCoA: Example Run

Let CoCoA solve a graph coloring problem, where each variable must be assigned value, which can be either blue (B), green (G), or yellow (Y) such that $\forall i D_i = \{\text{B}, \text{G}, \text{Y}\}$. The constraints in the graph coloring problem are that neighboring variables should not have the same color—a cost of one is induced for every pair of neighbors that are assigned the same color. In the example in Figure 1 the variables of A_1, A_2, and A_6 are already assigned a color. Let us assume that this is the starting condition and we have to find the best assignment in this situation starting at A_3. As we shall see under these premises the best solution must violate some constraints—there is no perfect solution with zero cost.

Agent A_3 starts by sending **UPDSTATE**$(3, \texttt{ACTIVE})$ and **INQMSG**$(3, \hat{\mathcal{X}}_3)$ to all of its single-hop neighbors $A_j \in \mathcal{M}_3 = \{A_1, A_2, A_4, A_5\}$, where $\hat{\mathcal{X}}_3 = \{X_1 = \text{G}, X_2 = \text{Y}\}$. As these messages arrive at all of the neighbors, they will save the information that is in the CPA and the state of A_3.

Each of the neighbors will then calculate a cost map Θ_j, which contains for every assignment $X_{i,k} \in D_i$, what the lowest possible local cost is, given the CPA (2). Agent A_1 will return a **COSTMSG**(Θ_1) message with the mapping $\Theta_1 = \{\text{G} \to 1, \text{Y} \to 0, \text{B} \to 0\}$ and for agent A_2 this mapping will be $\Theta_2 = \{\text{G} \to 0, \text{Y} \to 1, \text{B} \to 0\}$. For both A_4 and A_5 it will be $\Theta_4 = \Theta_5 = \{\text{G} \to 0, \text{Y} \to 0, \text{B} \to 1\}$, and agent A_3 its own costs are $\Theta_3 = \{\text{G} \to 1, \text{Y} \to 1, \text{B} \to 0\}$. All cost maps will be received by A_3, which sums over the possible assignments and finds $\{\text{G} \to 2, \text{Y} \to 2, \text{B} \to 2\}$. There are now three potential assignments leading to the same minimal cost. Since initially $\beta = 1$, the choice is delayed until more information is available (because $\beta < 3$ and other neighbors are still either ACTIVE or IDLE). Agent A_3 sends an **UPDSTATE**$(3, \texttt{HOLD})$ to its neighbors.

Since A_1 and A_2 are already DONE they will not react to the new information. Agents A_4 and A_5 are now activated and after inquiring their neighbors, they gather a combined mapping $\Theta_j = \{\text{G} \to 2, \text{Y} \to 2, \text{B} \to 1\}$. They can both find a unique minimal solution, so they assign their value to $X_j \leftarrow \text{B}$. Agents A_4 and A_5 send a **SETVAL**(j, B) to spread the algorithm's activity and their new value. Upon receiving this information the neighbors update their CPA accordingly. Immediately after, an **UPDSTATE**$(j, \texttt{DONE})$ message is sent, notifying all neighbors that they are now done.

Activity returns to A_3, who receives two **UPDSTATE**$(j, \texttt{DONE})$ messages. After the *first* message it will assume that another neighbor is still active and will run the algorithm again without increasing β. This time, A_3 will find assignment costs $\{\text{G} \to 2, \text{Y} \to 2, \text{B} \to 3\}$ (assuming that one of the neighbors is done, otherwise it would contain $\text{B} \to 4$; this is a race condition). As there are two minimizers and the uniqueness bound $\beta = 1$, A_3 will go to the HOLD state. After the second **UPDSTATE**$(j, \texttt{DONE})$ message arrives, A_3 knows that there are no more active neighbors, so it will increase its bound $\beta \leftarrow 2$. Now it will find the cost of assignments to be $\{\text{G} \to 2, \text{Y} \to 2, \text{B} \to 4\}$ with two distinct minima; since the uniqueness bound is now 2, it will select a random minimizer out of the two.

CoCoA: Termination Guarantees

With the state-mechanism in place, one could run the risk of entering an endless loop. We have the following Proposition:

Proposition 1. *The CoCoA algorithm will converge after a finite number of messages and function evaluations.*

Proof. Assume a situation in which an agent A_i and all of its neighbors are in the HOLD or DONE state. At some point A_i receives an **UPDSTATE**(j, S) message and it will find that there are no more active neighbors, thus increases its β by one. CoCoA will run again and either there will be a unique solution or not. If no solution is found, A_i will set its state to HOLD, we are again at an *impasse*, and the process will repeat. At some point however $\beta = |D_i|$ since the domain is finite. At this point any assignment must satisfy $\mathcal{U}(X_i) \leq \beta$, so a value will be picked, and an endless loop is avoided. □

CoCoA: Privacy

When solving ADCOPs there is always the possibility of transmitting the full local constraint cost matrix to one agent's neighbors. Sharing all constraint information between neighbors, and adding the received costs to the local costs, effectively converts any ADCOP into an equivalent symmetric DCOP. This strategy is also referred to as Private Events as Variables (Maheswaran et al. 2004), and the main motivation not to use this strategy is the loss of privacy.

Proposition 2. *CoCoA preserves at least as much privacy as the Max-Sum.*

Proof. The Max-Sum algorithm is known to be more privacy preserving than the local search algorithms ACLS and MCS-MGM (Zivan, Parash, and Naveh 2015). Only in two iterations entries from the cost matrices are exchanged between agents. In other iterations the shared values are derived from multiple entries as information spreads through the graph. CoCoA, on the other hand, shares cost matrix entries only once, i.e. before any agent has assigned a value

and after that it is always derived from multiple entries. In CoCoA, for each assignment, the lowest total cost is sent taking into account the complete CPA (line 16 of Algorithm 1). As with Max-Sum, in every message to A_i, $|D_i|$ values are transmitted, however since CoCoA does not iterate, the number of exchanged messages is lower. □

CoCoA Performance: Experimental Results

CoCoA is tested and compared against state of the art DCOP solvers, namely: DSA (Fitzpatrick and Meertens 2003) (variant C, with update probability, $p = 0.5$), MGM-2 (Maheswaran, Pearce, and Tambe 2004) (with offer probability $p = 0.5$), Max-Sum_ADVP (Zivan, Parash, and Naveh 2015) from hereon also referred to as simply Max-Sum, (switching graph direction after 100 iterations, value propagation after two switches, and using the constraint standard inner order), ACLS (with update probability $p = 0.5$) and MCS-MGM (Grubshtein et al. 2010) (non-parametric). Also we show the individual effects of one-step lookahead and the unique-first approach by showing the results for CoCoA with and without the unique-first (UF) strategy.

For all experiments 100 problems are generated (the type of problems will be described subsequently) and the presented results are the average over all problems. To compare the performance of CoCoA we look at the following performance metrics: (i) the cost of the final solution (S), (ii) the number of transmitted messages (M), (iii) the number of cost function evaluations (E), and (iv) running time of the algorithm (T). A cost function evaluation is defined as computing or looking up the local cost of one constraint, similar to Non Concurrent Constraint Checks (NCCCs) (Meisels et al. 2002). These are not necessarily non-concurrent but they do indicate a non-implementation specific measurement of computational effort. We keep track of the global cost function and when no better solutions are found for more than 100 iterations the solver is stopped. Afterwards we report the performance metrics at the moment where a solver was first within 1% of the best solution. This approach is similar to an anytime framework as described in (Zivan, Okamoto, and Peled 2014), but instead of keeping track of the best state at every agent, we maintain this information in the experiment script; this information is only used for evaluation.

The solvers are implementated in Java 1.7, and the experiments are set up in Matlab 2015b, which is also used to post-process and present the result figures[1]. The experiments are carried out on a laptop with an Intel Core i7-3720 CPU 2.6 GHz and 8 GB RAM.

Graph Coloring

Experiment Description A common problem for benchmarking DCOP solvers is the graph coloring problem e.g. (Maheswaran, Pearce, and Tambe 2004; Modi et al. 2005; Rogers et al. 2011). As in the example run, the values of $\mathcal{X}$ represent the colors of nodes, and the solvers need to assign colors such that nodes on the ends of edges have different colors. In the first experiment every constraint violation will induce a cost of 1. In the experiment the number of colors $|D| = 3$, so the cost matrix for every constraint is $\mathbf{C} = \mathbf{I}_3$. The graphs are generated by selecting $n = 500$ random points in two dimensional space using a Poisson point process, representing the variables, and the constraints are chosen as the edges of a Delaunay triangulation between those points—the average density of the graphs is 0.01.

[1]For a replicability of results and figures, the source code is available upon request, or at https://github.com/coenvl/mSAM.

Table 3: Graph coloring experiment results

Algorithm	I	S	M	E	T
CoCoA	N/A	183	**8827**	**137820**	**0.5**
CoCoA_UF	N/A	147	15402	151916	0.8
ACLS	35	189	101029	193588	3.5
DSA	200	**129**	28281	1176640	18.9
MCSMGM	58	144	127964	213273	6.0
MGM2	80	184	98236	451221	8.1

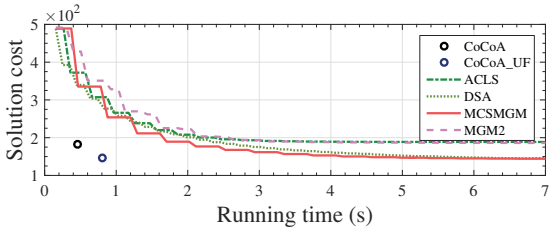

Figure 2: Graph coloring experiment: DSA finds the best solution, but CoCoA_UF finds a similar solution in less time.

Results In Figure 2 the solution cost is plotted against the running time for several algorithms, while Table 3 shows the full set of metrics. CoCoA_UF finds a solution of near optimal cost in a single iteration, requiring less function evaluations than any other algorithm, and second least number of messages; therefore it is also the fastest algorithm. The result demonstrates that the unique-first approach definitely provides a benefit in terms of eventual solution cost, as CoCoA_UF finds a solution that is 20% better than CoCoA at the cost of some additional messages and cost function evaluations, almost as good as DSA, which finds the best.

In the experiment Max-Sum is left out since it is unable to converge to a solution. This is because there are $|D|$ "mirrorred" solutions that perform equally well, and there is no local preference of one coloring over the other.

Semi-Randomized Asymmetric Problems

Experiment Description In the second experiment we generate semi-random asymmetric problems by creating scale-free graphs according to (Albert and Barabási 2002) with an initial graph of ten randomly connected nodes, and iteratively adding up to four nodes until $n = 200$, resulting in graphs with an average density of 0.04. The variable domain size $|D_i| = 10$, and for every constraint an integer semi-random cost is generated for both sides of the constraints. A cost of zero is selected with a probability of $p = 0.35$ and uniformly randomly chosen in the domain $[1, 100]$ for the remainder. This setup recreates the experiment as described

Table 4: Semi-randomized asymmetric experiment results

Algorithm	S	M	E	T
CoCoA	29316	**4813**	1595067	**0.3**
CoCoA_UF	27426	6873	1334186	**0.3**
ACLS	26550	153079	1295569	5.0
DSA	32337	40820	**1250377**	3.8
MCSMGM	**22200**	1349161	5396643	59.8
MGM2	35062	91805	3287152	11.5
Max_Sum	27077	1709434	51220868	133.9

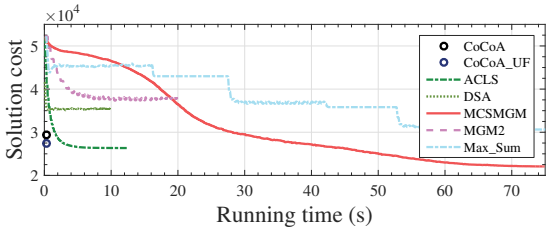

Figure 3: Semi-randomized asymmetric experiment: CoCoA finds a Pareto-optimal solution, but MCS-MGM eventually finds the best solution after more than 60 seconds.

Table 5: Sensor planning experiment results

Algorithm	S	M	E	T
CoCoA	**2365**	**1219**	388874	**0.1**
CoCoA_UF	**2365**	1763	331576	**0.1**
ACLS	3934	7018	**43516**	0.3
DSA	3332	13084	426799	1.3
MCSMGM	2379	197262	951732	10.7
MGM2	2917	24487	1161487	3.9
Max_Sum	1246306	12013	1299567	0.9

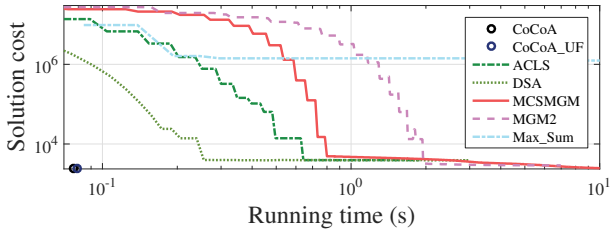

Figure 4: Sensor planning experiment: CoCoA finds the optimal solution and converges the fastest out of all evaluated algorithms.

in (Grinshpoun et al. 2013, Section 5.2).

Results Figure 3 presents the solution costs of different algorithms as they converge to a solution. CoCoA and CoCoA_UF quickly converge to a good solution, more than 10 times faster than any other algorithm. The symmetric DCOP solvers DSA and MGM-2 fail to find a solution. The Max-Sum algorithm also converges to a reasonable solution, but only after more than two minutes (not visible in Figure 3). The MCS-MGM algorithm shows a "discovery" phase in the first 25 seconds, after which it finds a global optimum, which is better than any other algorithm finds. The added overhead of the MCS-MGM algorithm can clearly be seen in Table 4. The better solution is found by sending nearly 200 times the amount of messages, evaluating 4 times the number of cost functions and running 200 times longer than CoCoA_UF.

Sensor Planning

Experiment Description The final experiment is motivated by an example in which a sensor network is used to monitor the cargo state of a shipping container. The sensors have to maintain a good quality estimation of the cargo such that they can either warn the cargo owner in case the shipping circumstances unexpectedly change, or provide a trace of the cargo state upon arrival. In this scenario the cargo estimation has to be optimized, but is constrained by limited battery life. The scenario is explained in more detail in (van Leeuwen et al. 2014), from which we use the outcome to model the effect of the communication frequency on the estimation quality, and on the battery lifetime. In this problem $\mathcal{X}$ are communication rates between sensors, and hard constraints make sure that the agents (nodes) will meet the minimum required battery life time. Asymmetric constraints between agents are used to model the effect that more *shared* information does not reduce the local estimation error, but it does improve the performance of neighboring nodes. The domains $\mathcal{D}$ are integer communication frequencies of $[1, 11]$ Hz, and the networks are generated by connecting $n = 50$ nodes, connected in randomly generated graphs with an average density of 0.17.

Results The results are show in Figure 4 and Table 5. They show that CoCoA is not only capable of finding the best solution, but also does so in the least amount of time and using the fewest messages from all considered algorithms—only in terms of computational effort, ACLS is more efficient. However, ACLS is unable to find a better solution than the symmetric solvers DSA and MGM-2. This may contribute to its low evaluation count—it is simply considered as "converged" after a small number of evaluations.

Conclusions

We have proposed and investigated a new ADCOP solver: CoCoA. We compared its performance with state of the art solvers by using (i) three-color graph coloring, (ii) randomized asymmetric problems and (iii) a sensor network use case problem. We showed that CoCoA finds high quality solutions, with a similar overhead for symmetric problems, and a much smaller overhead for asymmetric problems. The MCS-MGM algorithm found better solutions for randomized asymmetric problems, but required up to 200 times more messages, and over 4 times more cost function evaluations. In the sensor planning problems CoCoA finds better solutions, and does so faster than the benchmarks. We also conclude that preferring unique solutions whenever possible, yields a clear advantage in terms of solution cost.

Because CoCoA requires no iterative approach, it can be used in applications where fast convergence is required, or when the communication capabilities is a limiting factor. However, because it cannot recover from early choices, in the presence of hard constraints, it may not perform well.

The work is not complete—it is important to investigate how CoCoA performs under various circumstances, e.g. graph structures and densities, or different constraint functions.

References

Albert, R., and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74(1):47.

Chechetka, A., and Sycara, K. 2006. No-commitment branch and bound search for distributed constraint optimization. In *Proc. AAMAS*, 1427–1429.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. AAMAS*, 639–646.

Fitzpatrick, S., and Meertens, L. 2003. Distributed coordination through anarchic optimization. In *Distributed Sensor Networks*. Springer. 257–295.

Gershman, A.; Meisels, A.; and Zivan, R. 2009. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research* 34(1):61–88.

Grinshpoun, T.; Grubshtein, A.; Zivan, R.; Netzer, A.; and Meisels, A. 2013. Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research* 47:613–647.

Grubshtein, A.; Zivan, R.; Grinshpoun, T.; and Meisels, A. 2010. Local search for distributed asymmetric optimization. In *Proc. AAMAS*, 1015–1022.

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

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 *Proc. AAMAS*, 310–317.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2004. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proc. ICPADS*, 432–439.

Meisels, A.; Kaplansky, E.; Razgon, I.; and Zivan, R. 2002. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS*, 86–93.

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

Modi, P. J. 2003. *Distributed Constraint Optimization For Multiagent Systems*. Ph.D. Dissertation, University of Southern California, Los Angeles, CA, USA.

Pearce, J. P., and Tambe, M. 2007. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proc. IJCAI*, 1446–1451.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *Proc. IJCAI*, 266–271.

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

van Leeuwen, C. J.; de Gier, J. M.; de Filho, J. A. O.; and Papp, Z. 2014. Model-based architecture optimization for self-adaptive networked signal processing systems. In *Proc. SASO*, 187–188.

Yedidsion, H.; Zivan, R.; and Farinelli, A. 2014. Explorative max-sum for teams of mobile sensing agents. In *Proc. AAMAS*, 549–556.

Yeoh, W., and Yokoo, M. 2012. Distributed problem solving. *AI Magazine* 33(3):53–65.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowl. Data Eng.* 10(5):673–685.

Zivan, R., and Peled, H. 2012. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *Proc. AAMAS*, 265–272.

Zivan, R.; Okamoto, S.; and Peled, H. 2014. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence* 212:1–26.

Zivan, R.; Parash, T.; and Naveh, Y. 2015. Applying max-sum to asymmetric distributed constraint optimization. In *Proc. IJCAI*, 432–438.