Hybrid DCOP Solvers: Boosting Performance of Local Search Algorithms

Cornelis Jan van Leeuwen[1,2] and Przemyzław Pawełczak[2]

[1] TNO, Eemsgolaan 3, Groningen, The Netherlands
coen.vanleeuwen@tno.nl
[2] Delft University of Technology, Van Mourik Broekmanweg 6, Delft, The Netherlands
{p.pawelczak,c.j.vanleeuwen-2}@tudelft.nl

Abstract. We propose a novel method for expediting both symmetric and asymmetric Distributed Constraint Optimization Problem (DCOP) solvers. The core idea is based on initializing DCOP solvers with greedy fast non-iterative DCOP solvers. This is contrary to existing methods where initialization is always achieved using a random value assignment. We empirically show that changing the starting conditions of existing DCOP solvers not only reduces the algorithm convergence time by up to 50%, but also reduces the communication overhead and leads to a better solution quality. We show that this effect is due to structural improvements in the variable assignment, which is caused by the spreading pattern of DCOP algorithm activation.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) are a method for formalizing and solving problems that have a distributed nature, and in which multiple cooperating agents control discrete variables in order to optimize a common problem. DCOPs can be found in many different domains such as sensor networks [5], mobile sensing team coordination [25], communication [26], home automation [21] and smart grid optimization [8]. The underlying structure of DCOPs is always the same: agents have to assign variables that not only optimize a local set of constraints, but have to send messages to other agents in order to cooperatively come up with variable assignments that are optimal for the complete set of agents. A special kind of DCOPs can be formulated where agents having a shared constraint may assign different costs for a value assignment. These problems are called Asymmetric DCOPs (ADCOPs) [10]. In ADCOPs the aspect of cooperation is even more important, since an assignment leading to a local improvement may deteriorate the global performance.

A variety of algorithms that find solutions for DCOPs can be classified as either *complete* or *incomplete* solvers [12]. Solvers such as ADOPT [17], DPOP [19] or AFB [9] are the algorithms of the complete type, and are used to find the optimal solution. However, DCOPs are NP-hard [16] so a solution becomes intractable for large scale problems. Therefore incomplete solvers use heuristic

approaches to find a solution which may be suboptimal, but reaching the solutions much faster. In other words, there is a trade-off between solution quality and speed.

In this paper we introduce a new class of incomplete DCOP solvers, that combine features of different DCOP solvers and combine them into *hybrid solvers*. Specifically, we show that we can use different *initialization* methods for existing DCOP algorithms, which has a profound impact on their overall performance. In order to do so we show that using different initialization methods that are *not* iterative in approach, and are hence very fast in converging to a solution, we can reduce algorithm running times and improve the solution quality.

In the evaluation of the proposed hybrid DCOP solvers, we will consider existing symmetric and asymmetric solvers, as well as a new algorithm which is an extension of ACLS, which we will refer to as ACLS-UB.

2 Problem Statement and Notation

Before introducing our proposed novel class of hybrid DCOP solver we need to start with providing the definition of (A)DCOPs. DCOPs are problems from the field of multi-agent systems in which agents can reason and send messages to one another to cooperatively decide on their variable assignments in order to find a solution to a global cost minimization function.

Problem Formalization and Notation Following the notation from [9], DCOPs are defined as a tuple $\mathcal{T} = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$ and $\mathcal{X}$ is the set of variables $\{X_1, X_2, \ldots, X_n\}$ with finite discrete domains $\{D_1, D_2, \ldots, D_n\}$ from the set of domains $\mathcal{D}$ such that $X_i \in D_i$. Furthermore we require that each agent A_i is assigned one corresponding variable, X_i, and therefore $|\mathcal{A}| = |\mathcal{X}| = |\mathcal{D}|$. This one-on-one relation between agents and variables, is seen in many DCOP studies, but is not strictly required. Then, $\mathcal{R}$ is a set of relations or constraints between variables, in which each constraint $C \in \mathcal{R}$ defines a non-negative cost depending on the value assignment of the involved variables. Every possible value assignment of a set of variables has a particular induced global cost $C\colon\ D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0}$, while for ADCOPs each constraint defines a set of costs for every 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, as for most DCOP studies, we shall only take into account binary constraints, in which exactly two variables are involved for every constraint, which is then 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 exists a constraint between the corresponding variables. This follows the real-life situation of limited range

between cooperative 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 depicted as an undirected graph.

3 A New Class of DCOP Solvers: Hybrid Solvers

In this paper we propose a new class of DCOP solvers in which we propose a simple, yet an effective idea. We take the best of different types of existing DCOP solvers, forming a new class of *hybrid* DCOP solvers. Particularly, we propose to improve the performance of existing local search algorithms by modifying the initialization methods to find the initial value assignment. In the field of Constraint Satisfaction Problems, which is closely related to DCOPs, the approach of using a initialization and repairing it is well known, and can yield great benefits [15]. From the fields of evolutionary algorithms [20], clustering [2], neural networks [24, 4] and meta-learning [7] we know that initialization methods can have a great effect on the performance of an algorithm. However to the best of our knowledge, there has been little to no work on the effects of different initialization methods for (A)DCOPs. In this paper we will study the effect different initialization methods may have on the performance of (existing) DCOP algorithms.

Hybrid DCOP Solver: We define a hybrid DCOP solver as a solver which executes sequentially other (existing) DCOP solvers. Selection of *which* DCOP method to use in the next DCOP solving iteration and *when* to switch to a new DCOP solver method is a core of the hybrid solver definition.

3.1 Motivation for a Hybrid DCOP Solver

Most (if not all) local search DCOP algorithms use an initial random assignment for all of the variables, which will be iteratively improved upon. Instead, an initial assignment can be computed by a non-iterative DCOP algorithm, such as a simple greedy algorithm, or a more elaborate greedy algorithm such as the one introduced in [22]. Since these methods will assign a value only once, and then terminate, they quickly provide a good initialization assignment from which one can start another DCOP method.

We hypothesize that the combination of different initialization methods for iterative algorithms in DCOP solution search, will be beneficial because of two effects:

1. *Solution quality improvement over initial assignment*: most probably a simple initialization method will find a sub-optimal solution, and many local search algorithms will be able to improve it. Algorithms that are known to provide monotonically decreasing solution costs (any algorithm that uses a

coordinated change approach, e.g. MGM-2, ACLS, MCS-MGM) are guaranteed to find better (or equal) solutions compared with the initial value assignment; and
2. *Increased convergence speed for local search algorithms*: DCOP algorithms that use local search will most likely converge faster when a good solution is used for initial DCOP value assignment. This will lead to a shorter total running time for algorithms that are initialized with a better assignment.

4 Initialization of DCOP Solvers: Classification

In our experiments we combine different *initialization methods* with existing *local search algorithms* which we shall also refer to as *iterative methods*. Since the aim of this study is to improve the solution quality and convergence speed of solvers, we do not take into account complete solvers. To understand why only certain combination of DCOP solvers improves the solution, we need first to classify (i) initialization methods, and (ii) types of DCOP iterative solvers.

4.1 DCOP Classification: Initialization Methods

- **Random** A de facto standard method for all DCOP solvers. It does not take into account any constraints and starts with the random variable assignment.
- k **Step Look-ahead:** We define a look-ahead initialization algorithm as the one in which one randomly chosen initial node is triggered first, and only after it has chosen a value it will activate its neighbors. When choosing a value, it takes into consideration the effect on all of its neighbors that are reachable within k steps (edges or hops). Three special cases of k step look-ahead are already known and described in the literature:
  - **Zero Step Look-ahead (ZSLA):** A zero step look-ahead algorithm ($k = 0$) is the one in which an agent optimizes only for the constraints it is directly involved in. Such algorithm are also referred to as *greedy, breadth-first* algorithms;
  - **Single Step Look-ahead (SSLA):** A single-step-look-ahead algorithm ($k = 1$) is defined as the one in which an agent optimizes not only for the constraints it is involved in, but also the constraints its one-hop neighbors are in. One such algorithm is the recently proposed CoCoA algorithm [22] and its variants CoCoA_UF and CoCoA_WPT [23];
  - **Max Step Look-ahead (MSLA):** If k is equal to the height of the graph's minimal spanning tree, and the algorithm would be started at the root of the spanning tree, the algorithm becomes a complete algorithm, and is in fact equivalent to DPOP [19].

4.2 DCOP Classification: Existing Iterative Methods

Classifying iterative methods used in DCOP solvers we can divide them into two main groups:

- **Symmetric DCOP Solvers:** Those include DSA [27], MGM and MGM2 [13] and generalized DBA [18];
- **Asymmetric DCOP Solvers:** Those include MCS-MGM [11], and ACLS [10] with its new version ACLS-UB (which is also a novel contribution of this work and described in Section 4.3).

Remark on Max-Sum: We are naturally aware of another popular DCOP solver: the Max-Sum algorithm [6] or any one of its variants. However Max-Sum is unable to utilize the benefit of initializing, as it tries to approximate the global utility of any value, and uses this to determine the best variable assignment. There are extensions of Max-Sum that are able to build upon an an initial assignment by using value propagation [29]. In a recent paper [3] the effect of initialization was studied in variant called Max-Sum_ADSSVP. The authors find that the timing, and approach to initialization has a great effect on the performance, both in terms of solution quality and convergence speed. For our evaluation however, we leave Max-Sum out of the comparison, and refer to this paper to provide a complete overview of different hybrid algorithms.

4.3 Novel Iterative DCOP Solver: ACLS-UB

In addition to existing DCOP algorithms listed above we introduce a variant of the ACLS, denoted as *Unbiased* (ACLS-UB).

ACLS-UB Algorithm: In the original ACLS algorithm [10], at every iteration an agent chooses a variable assignment that would lower its local costs and proposes it as a new value to its neighbors. Neighbors respond with the effect on their side, after which the proposition which has the best effect on the regional cost function is selected. In the ACLS-UB algorithm a value assignment is proposed from *all* possible values, instead from the subset that improves its local state. The ACLS-UB algorithm is described using pseudo code in Algorithm 1.

ACLS-UB works by iteratively proposing a random value from its domain D_i, and sends that value to its neighbors. The neighbors respond by sending the effect of the assignment on their local costs, taking into account all known value assignments. When these local effects are received by the initial agent, it sums over all received effects, and assigns the proposed value with probability p only if it will reduce the current local cost.

Relation of ACLS-UB to Other Solvers: The main difference between ACLS and ACLS-UB is in line 4 of Algorithm 1, where any random value is picked from the domain. In the long run, the effect of this pick is that the effect of all values from the domain are used to retrieve the induced effect on the neighbors' local cost.

Intuitively ACLS-UB works very similar to CoCoA [22, 23], with the major difference that CoCoA operates in one single iteration instead of iteratively trying different values. Another difference is that in ACLS-UB the neighbors will send back the value of the constraint cost, whereas CoCoA will send back the

Algorithm 1 ACLS-UB Algorithm

On A_i when activated
$X_i \leftarrow$ chooseRandomValue()
while (no termination condition is met) **do**
  send X_i to $\forall A_j \in \mathcal{M}_i$
  $\upsilon \leftarrow$ chooseRandomValue()
  send υ to $\forall A_j \in \mathcal{M}_i$
  wait for incoming constraint cost δ_j from A_j
  $\Delta_i \leftarrow \sum_j^{j \in \mathcal{M}_i} \delta j$
  if $\Delta_i <$ currentCost **and** random$[0, 1] < p$ **then**
    assign $X_i \leftarrow \upsilon$
  end if
end while

On A_j when receiving υ from A_i
send constraint cost δ_j for $X_i = \upsilon$

lowest induced cost for any assignment in conjunction with the proposed value and the CPA. This extra look-ahead is not efficient in ACLS-UB, since the next-hop neighbors *will* in fact already have an assignment, and the lowest cost will be too optimistic. Note that the unique-first approach of CoCoA is not required in ACLS-UB, as it can easily recover from any earlier suboptimal assignments in later iterations, whereas CoCoA cannot.

5 Hybrid DCOP Solvers: Introduction and Initial Results

In order to understand whether there is any benefit from hybrid solvers, we performed the following experiments[3]. For any problem, we initiate 200 problem instances which are initialized by three methods: *random*, *ZSLA* (i.e. greedy) and *SSLA* (i.e. CoCoA) and subsequently solved by other DCOP solvers (depending on the experiment). We report the average result of all problems. We assume a solver has converged when no better solutions have been found for more than 100 iterations, and define the moment of "convergence" as the first iteration in which the solution was within 1% of the minimal solution. In this way we can compare the convergence speed of different algorithms, and do not have to specify the number of iterations beforehand, in a way similar to the any-time solution as proposed in [28].

As performance metrics we will score solvers on the following metrics:

- **The number of iterations required to converge**, denoted as I;
- **Final cost of the solution after the algorithm converged**, denoted as S;

[3] For reproducibility and validation of our results, all code for the algorithms is available at https://github.com/anonc187/jCoCoA, and for the experimental setups at https://github.com/anonc187/mCoCoA.

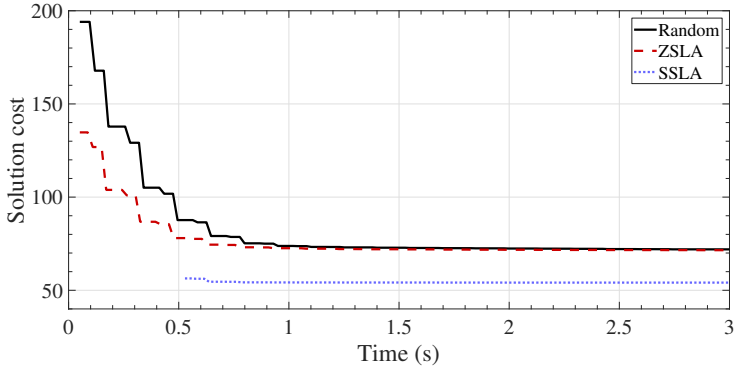

Fig. 1. In a three-color graph coloring experiment, when using an SSLA for initialization, the MGM-2 algorithm has a great benefit in speed and solution quality.

Table 1. Graph coloring experiment results

Algorithm	I	S	M*	E*	T
Random_DSA	157	49	10.4	362.9	3.5
ZSLA_DSA	164	49	10.7	381.6	3.7
SSLA_DSA	115	47	**14.5**	381.3	3.1
Random_MGM2	55	72	26.3	120.2	1.7
ZSLA_MGM2	42	71	21.0	94.4	1.3
SSLA_MGM2	7	54	12.2	121.0	0.7

$* = \times 10^3$

- **Number of messages that are transmitted during the run**, denoted as M;
- **Number of constraint evaluations**, denoted as E; and
- **Running time until the moment of convergence**, denoted as T in seconds.

The constraint evaluations are indicative of the computational complexity and can also be referred to as Non Concurrent Constraint Checks (NCCC)s [14].

5.1 Experiment Results

Experiment 1: Symmetric DCOP We use a (symmetric) graph-coloring problem with three colors, which have to be assigned to 200 variables. The constraints between the variables are chosen as the nodes were connected via a Delaunay triangulation, when the variables are points chosen randomly on a two-dimensional plane. The results of MGM-2 ($p = 0.5$) is shown in Figure 1, of which the numeric results are shown in Table 1 together with DSA (variant C, with $p = 0.5$).

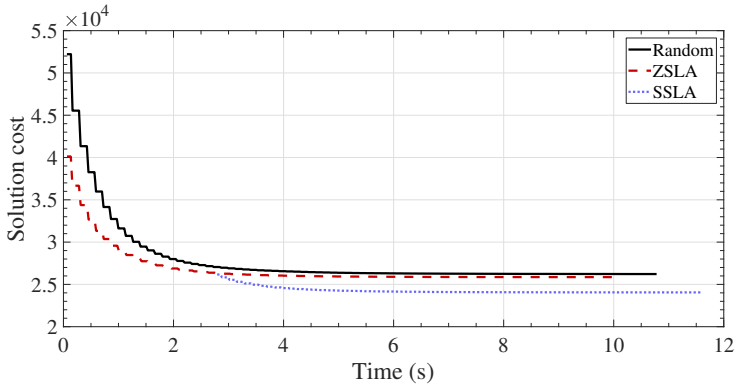

Fig. 2. In a semi-random experiment (Section 5.1, Experiment 2), when using an SSLA for initialization, the ACLS algorithm shows faster convergence to a better solution.

Table 2. Semi-randomized asymmetric experiment results

Algorithm	I	S*	M*	E*	T
Random_ACLS	95	26.4	151	1321	4.3
ZSLA_ACLS	75	26.1	121	995	3.5
SSLA_ACLS	48	24.2	107	10571	**4.9**
Random_ACLSUB	333	24.7	531	5747	14.8
ZSLA_ACLSUB	299	24.1	477	5164	13.3
SSLA_ACLSUB	207	23.5	358	12963	11.9
Random_MCSMGM	1154	21.7	1389	5557	55.1
ZSLA_MCSMGM	1022	21.6	1232	4935	48.8
SSLA_MCSMGM	781	22.0	970	13599	40.0

$* = \times 10^3$

Experiment 2: Asymmetric DCOP An asymmetric problem is chosen where the constraints are created using a scale-free graph generation method [1], and are assigned semi-random asymmetric costs such that there is a high probability that a conflict of interests occurs. This problem is created specifically to benchmark asymmetric problems, and is described in more detail in [10, Section 5.2]. The result of the ACLS ($p = 0.5$) algorithm is shown in Figure 2, and the results of all algorithms is presented in Table 2.

Hybrid Solvers—Discussion of Initial of Results: Based on the results from the first two experiments, we see that local search DCOP solvers reduced their execution time and communication overhead using initialization methods other than random, but surprisingly also found a final solution with a lower cost. The only two exceptions (marked in bold in Table 1 and 2) are (i) when using the SSLA with DSA, in which case the messages of the SSLA increases the very

low communication overhead of DSA, and (ii) when using an SSLA with ACLS, in which case the added run time of the SSLA increases the convergence time. The speed performance gain can be easily explained: a better initialization will reduce the amount of variable "tweaking" needed. However, how come the final solution is also *lower* and solution result dependent on the initialization method?

5.2 Why Do Only Some Hybrid DCOP Solvers Work?

We introduce three hypotheses to explain this phenomenon:

1. **Hypothesis 1:** A lower initial solution will always lead to a lower final solution;
2. **Hypothesis 2:** Using initialization methods other than random increases the explored portion of the solution space;
3. **Hypothesis 3:** The initialization method itself finds a starting point in the search space, from which a lower local minimum is reachable.

Let us experimentally verify these three hypotheses in detail.

Verifying Hypothesis 1: Solution Cost Correlation The SSLA algorithm finds a lower initial cost than the ZSLA initializer, which in turn finds a lower cost than a random assignment. Hence the first hypothesis is that a lower initial costs will (on average) lead to lower final costs. The initial state is known to be of great influence on the final solution, and a correlation between the initial cost and the final cost could explain why ZSLA or SSLA initialization methods lead to better final solutions.

To test this hypothesis we performed an experiment by repeatedly invoking the algorithms on the exact same problem setup, but with different random initializations. We gather information on the cost at initialization and of the eventual outcome. In Figure 3 we show the minimum, average, and maximum results of 200 instantiations of the algorithms solving the exact same three color graph coloring problem with a Delaunay graph of size $n = 100$. For this small experiment we only compare the DSA algorithm instantiated with a randomized approach with an algorithm that uses CoCoA_WPT for initialization.

From this experiment we see that for some iterations we *do* find a solution with the random strategy which is as good as the SSLA-initialized solution, however on average the final solution is worse. The spread of the initial and the final solution corresponds with the statistical spread of the random assignments, and some random initialization lead to better final solutions than others. If we look at the correlation between the initial cost and the final cost of the individual runs, then we find that there is no correlation between the cost of the initial random assignment and the final minimal cost. The Pearson correlation coefficient between the solution cost at the beginning and the end is 0.15. With these results we *reject hypothesis 1.*

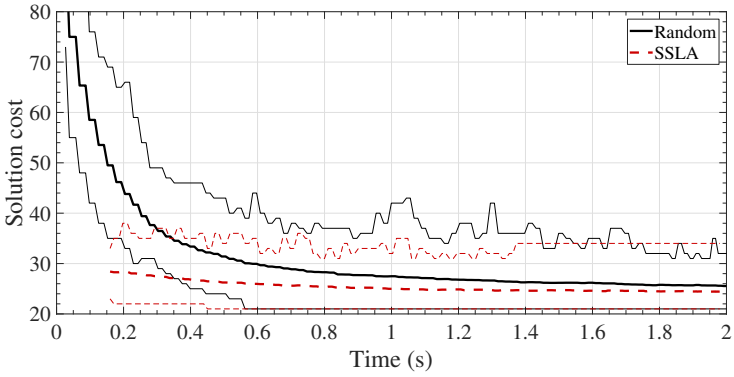

Fig. 3. Starting the DSA algorithm from a different starting point will lead to a different outcome. This graph shows the minimum, average and maximum solution costs during the experiments.

Verifying Hypothesis 2: Increased Solution Space Exploration A DCOP problem is generally a matter of solution space exploration. The solvers that are capable of effectively searching a larger portion of the solution space, will reasonably find a better final solution than solvers that cannot. Since local search algorithms search only a small fraction of the search space, the increase in search space exploration by a SSLA may be of large influence. Put differently, a better overview of the trends in the solution space may lead to insights as to where the best optimum lies. If we can show that the solvers using SSLA explore a larger part of the solution space than randomly initialized solvers, this may explain why they find solutions with the lower final cost.

To verify this, we construct an experiment in which we captured the CPA **every** time a constraint check is performed, so that we can store every explored value assignment. We did observe that SSLA searches a slightly larger portion of the solution space than ZSLA, which in turn searches a larger part of the solution space than random. However the SSLA algorithm sometimes searches a **smaller** part of the solution space, largely because its successor algorithm converges so quickly. Therefore, with these results we *reject hypothesis 2* as well, also because the differences are so marginally small that they cannot explain the significant effect on the results.

Verifying Hypothesis 3: Selection of Starting Point The initial assignment determines the area of the solution space that is reachable through local search. It is possible that SSLA is capable of finding initial assignments that have relatively good local minima. To explain what we mean by this, let us sketch the following example.

Definition 1. *A **bridge edge** is defined as an edge that, when removed from the graph, the graph will no longer be connected.*

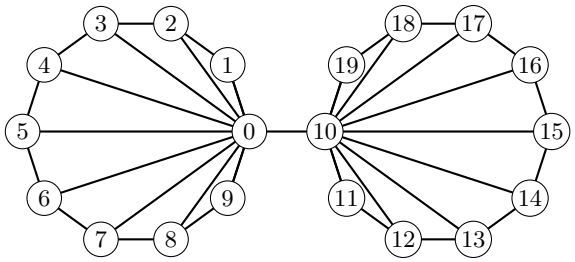

Fig. 4. An example graph in which two dense clusters of nodes are connected by a single bridge.

Suppose we have a graph with high modularity, meaning it consists of clusters of densely connected nodes, which are connected through relatively low number of *bridge* edges. The nodes on these bridges could initially induce a high performance penalty; and it may be impossible for a local search algorithm to escape from these expensive assignments because of the many low cost constraints around it on the surrounding nodes.

As a minimal example suppose we have a three color graph coloring problem with a graph as shown in Figure 4. As we see node 0 and 10 have a constraint with many other nodes, and are connected to one another. If through some unfortunate random assignment, they are both given the same initial color (for example red) and the nodes around it are mostly other colors, then no local search algorithm will change that initial assigned color. This is confirmed through experiments in which we use the graph as depicted in Figure 4, letting the algorithms solve the graph-coloring problem. One group of agents is hardwired to intialize with an initialization in which $X_0 = X_{10}$, and $\forall_{i \neq 10} X_i \neq X_0$. As we can see in Figure 5, the local search algorithm is unable to find a solution in which this constraint is resolved. However, we can guarantee that an SSLA algorithm will never assign the same color to endpoint vertices of a bridge, and will thus lead to better solutions.

Proposition 1. *A SSLA will never assign the same color to the endpoints of a bridge.*

Proof. (*Sketch*) When an SSLA starts, any random agent is activated first. If either bridge endpoint is selected first, then logically they will not be activated simultaneously. If any other node is selected, then this node will execute the algorithm and select a random initial assignment. After that it activates all of its neighbors, which will execute the algorithm, until at some iteration the first bridge endpoint is selected. At this moment the other endpoint cannot be activated, or the edge would not have been a true bridge.

Because the algorithm is active in one bridge endpoint, but never in both at the same time, one must assign a value before the other. Moreover, when an endpoint of the bridge eventually has to assign a value, no nodes from the other component can be assigned a value yet, because the bridge is the only connecting

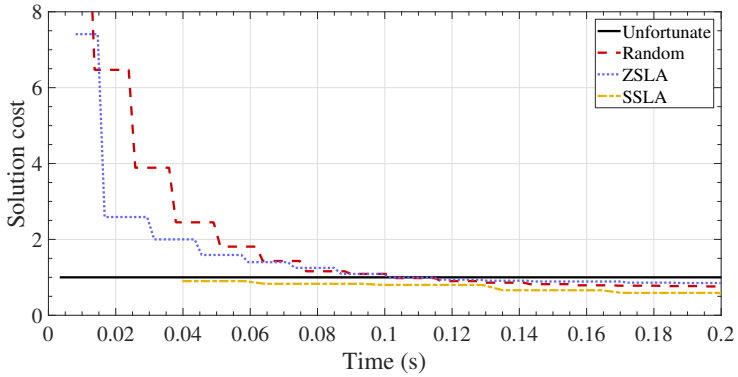

Fig. 5. The results of the MCS-MGM algorithm trying to solve the graph coloring problem in graph from Figure 4, with various initialization strategies. The "*unfortunate*" strategy is hardwired to get a conflict on the bridge.

edge. Therefore when the second bridge endpoint is activated it will only have the first endpoint as a constraining value, and will thus always pick a different value.

Although this exact order of events will not hold for pseudo-bridges that connect clusters within a graph, there will be an ordering in which the nodes will be activated, as long as the detour path between the vertices on the pseudo-bridge is longer than three. In many graphs with high modularity, the values of nodes in the bridging constraints will therefore be chosen with low costs, and the coloring within the clusters can simply be permuted. Therefore the local search algorithms that continue from these solutions are generally of higher quality, than those from random initial assignments.

If this final hypothesis is true, then we expect some different results in the performance of different types of graphs, especially for various densities. We would expect the benefit of an SSLA to decrease with problem graphs of higher densities, since in these graphs bridges or pseudo-bridges occur less frequently.

6 Graph Density

In our final experiment we use once more the graph coloring problem with $|D| = 3$, and instantiate randomly connected graphs with $n = 200$ with nine varying densities between 0.01 and 0.3. We generate 50 graphs of every density, let the different solver combinations (initialization and iteration) solve the same graphs, and report the average performance of all instances. The convergence criteria were identical to the experiment described in Section 5.

In Figure 6 we show the averaged results for the MGM-2 solver, when solving the graphs with different densities. We can indeed conclude that for graphs with

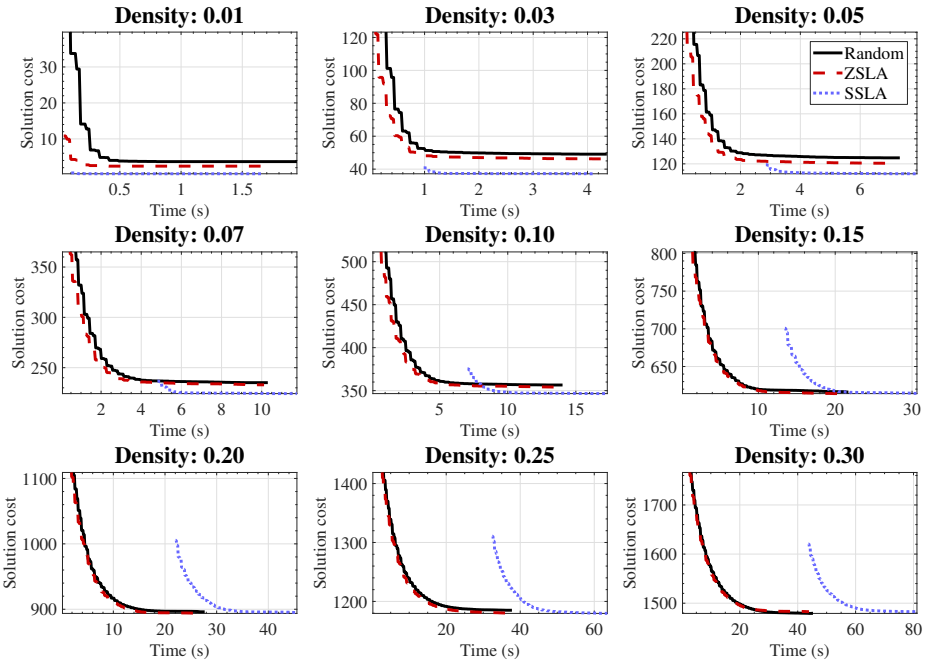

Fig. 6. The solution cost versus the time of the MGM-2 algorithm when solving random graphs of different densities shows the impact of initialization methods. Up to a density of 0.1 there is a clear improvement on the solution cost.

a low to medium density (up to 0.1), there is a benefit using SSLA initialization for the final solution cost. The increase in convergence speed deteriorates much faster, since the complexity of the SSLA is exponential with the node degree, and this increases with graph density. Note, the time the SSLA initialization takes is shown as the starting point of the line.

For other local search algorithms (DSA, ACLS, MCS-MGM), similar results were found.

7 Discussion

In this article we studied the effect of different initialization strategies on the performance of different DCOP algorithms. Particularly, we introduced a new class of hybrid algorithms which combine the strategies of SSLA algorithms with iterative local search algorithms. We found that using this combination not only combines the fast convergence of the SSLA with the eventual better solution quality of the iterative approach, but that using the hybrid solver actually improves the quality of the final solution compared to using the iterative approach alone.

Two possible hypotheses that could explain this observation were rejected: (i) better initializations do not necessarily lead to lower final solution costs, and (ii) using SSLA does not significantly increase the searched solution space. Instead we hypothesize that using an SSLA (such as CoCoA) selects an initialization that is in a region of the solution space which has a lower local minimum than the statistical expected local minimum. This is caused by a reduction of conflicting values assigned on bridge vertices. In our final experiment we show that the effect is most abundant on low density graphs, in which (pseudo-)bridges are more present, and the solution cost of the search space is less homogenous.

Our hybrid approach seems well suited for applications in which maximum performance in terms of both convergence speed and solution cost is required. In fact we may use it as a general strategy for initial value assignment instead of using random values in problems with low graph densities.

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47–98 (2002)
2. Cao, F., Liang, J., Jiang, G.: An initialization method for the k-means algorithm using neighborhood model. Computers & Mathematics with Applications **58**(3), 474–483 (2009)
3. Chen, Z., Deng, Y., Wu, T.: An iterative refined Max-sum_AD algorithm via single-side value propagation and local search. In: Proc. AAMAS. pp. 195–202 (May 8–12 2017)
4. Dolezel, P., Skrabanek, P., Gago, L.: Weight initialization possibilities for feedforward neural network with linear saturated activation functions. In: Proc. IFAC. vol. 49, pp. 49–54. Brno, Czech Republic (Oct 5–7 2016)
5. Farinelli, A., Rogers, A., Jennings, N.R.: Agent-based decentralised coordination for sensor networks using the max-sum algorithm. Autonomous Agents and Multi-Agent Systems **28**(3), 337–380 (May 2014)
6. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proc. AAMAS. pp. 639–646. Estoril, Portugal (May 12–16 2008)
7. Feurer, M., Springenberg, J.T., FrankHutter: Initializing bayesian hyperparameter optimization via meta-learning. In: Proc. AAAI. pp. 1128–1135 (Jan 25–03 2015)
8. Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., Ranade, S.: A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In: Proc. AAMAS. pp. 999–1007 (May 8–12 2017)
9. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding for distributed COPs. Journal of Artificial Intelligence Research **34**(1), 61–88 (Jan 2009)
10. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., Meisels, A.: Asymmetric distributed constraint optimization problems. Journal of Artificial Intelligence Research **47**, 613–647 (May 2013)
11. Grubshtein, A., Zivan, R., Grinshpoun, T., Meisels, A.: Local search for distributed asymmetric optimization. In: Proc. AAMAS. pp. 1015–1022. Toronto, Canada (May 10–16 2010)
12. Leite, A.R., Enembreck, F., Barthès, J.P.A.: Distributed constraint optimization problems: Review and perspectives. Expert Systems with Applications **41**(11), 5139–5157 (Sep 2014)

13. Maheswaran, R.T., Pearce, J.P., Tambe, M.: Distributed algorithms for DCOP: A graphical-game-based approach. In: Proc. ICPADS. pp. 432–439. San Fransisco, CA, USA (Sep 15–17 2004)
14. Meisels, A., Kaplansky, E., Razgon, I., Zivan, R.: Comparing performance of distributed constraints processing algorithms. In: Proc. AAMAS. pp. 86–93. Bologna, Italy (Jul 15–19 2002)
15. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence **58**(1-3), 161–205 (Dec 1992)
16. Modi, P.J.: Distributed Constraint Optimization For Multiagent Systems. Ph.D. thesis, University of Southern California, Los Angeles, CA, USA (2003)
17. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence **161**(1–2), 149–180 (Jan 2005)
18. Okamoto, S., Zivan, R., Nahon, A.: Distributed breakout: Beyond satisfaction. In: Proc. IJCAI. pp. 447–453. New York, NY, USA (Jun 9–15 2016)
19. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proc. IJCAI. pp. 266–271. Edinburgh, Scotland, UK (July 30–Aug 5 2005)
20. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: A novel population initialization method for accelerating evolutionary algorithms. Computers & Mathematics with Applications **53**(10), 1605–1614 (2007)
21. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: Proc. IJCAI. pp. 468–474. New York, NY, USA (Jul 9–15 2016)
22. van Leeuwen, C.J., Pawełczak, P.: CoCoA: a non-iterative approach to a local search (A)DCOP solver. In: Proc. AAAI. pp. 3944–3950. San Fransisco, CA, USA (Feb 4–11 2017)
23. van Leeuwen, C.J., Yıldırım, K.S., Pawełczak, P.: Selfadaptive safe provisioning of wireless power using DCOPs. In: Proc. SASO 2017. Tucson, AZ, USA (Sep 18–22 2017)
24. Yam, J.Y., Chow, T.W.: A weight initialization method for improving training speed in feedforward neural network. Neurocomputing **30**(1), 219–232 (2000)
25. Yedidsion, H., Zivan, R., Farinelli, A.: Explorative max-sum for teams of mobile sensing agents. In: Proc. AAMAS. pp. 549–556. Paris, France (Jan 5–9, 2014)
26. Yeoh, W., Yokoo, M.: Distributed problem solving. AI Magazine **33**(3), 53–65 (2012)
27. Zhang, W., Wang, G., Zhao, X., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence **161**(1), 55–87 (Jan 2005)
28. Zivan, R., Okamoto, S., Peled, H.: Explorative anytime local search for distributed constraint optimization. Artificial Intelligence **212**, 1–26 (Jul 2014)
29. Zivan, R., Peled, H.: Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In: Proc. AAMAS. pp. 265–272. Valencia, Spain (Jun 4–8 2012)