Improving Approximate Algorithms for DCOPs Using Ranks

Andreas Flueckiger, Mihaela Verman, and Abraham Bernstein

Department of Informatics, University of Zurich,
Binzmuehlestrasse 14, 8050 Zurich, Switzerland
andreas@flueckiger.ch,{verman,bernstein}@ifi.uzh.ch
http://www.ifi.uzh.ch

Abstract. Distributed Constraint Optimization Problems (DCOPs) have long been studied for problems that need scaling and are inherently distributed. As complete algorithms are exponential, approximate algorithms such as the Distributed Stochastic Algorithm (DSA) and Distributed Simulated Annealing (DSAN) have been proposed to reach solutions fast.
Combining DSA with the PageRank algorithm has been studied before as a method to increase convergence speed, but without significant improvements in terms of solution quality when comparing with DSA. We propose a modification in terms of the rank calculation and we introduce three new algorithms, based on DSA and DSAN, to find approximate solutions to DCOPs.
Our experiments with graph coloring problems and randomized DCOPs show good results in terms of solution quality in particular for the new DSAN based algorithms. They surpass the classical DSA and DSAN in the longer term, and are only outperformed in a few cases, by the new DSA based algorithm.

Keywords: Distributed Constraint Optimization Problems, PageRank

1 Introduction

Constraint Optimization Problems (COPs) are important in areas such as resource allocation. A Distributed Constraint Optimization Problem (DCOP) is a COP that distributes the constraints and variables among several agents. That can be desirable to solve large problems or to avoid the disclosure of all information to a single agent.

COPs and DCOPs can be solved with complete algorithms, which guarantee to find an optimal solution. However, COPs are NP-hard and complete algorithms can easily exceed any reasonable time limit.

Incomplete algorithms such as the Distributed Stochastic Algorithm (DSA) [17] aim to find good solutions within a short time. Other algorithms are often benchmarked against DSA. Verman et al. [13] developed a modified version of DSA with the goal of outperforming the original DSA in terms of speed of convergence. Their algorithm uses a modified PageRank [10] algorithm and is called Ranked DSA (RDSA). RDSA performs well in the beginning, but it easily converges to a local optimum of low quality compared to the original DSA.

This paper proposes a further modification of RDSA to overcome that problem. It also applies the new modification to the Distributed Simulated Annealing (DSAN) algorithm [1], which inherently allows for more exploration. We also introduce a generalization of the ranked algorithms for randomized DCOPs, since the original RDSA was designed specifically for graph coloring problems.

The performance of the new and existing algorithms is compared in experiments, using graph coloring problems and randomized DCOPs.

2 Related Work

2.1 Distributed Constraint Optimization Problems

Corresponding to [16], a Constraint Satisfaction Problem (CSP) consists of n variables $v_1 \dots v_n$ that take their values from the domains $D_1 \dots D_n$, and a set of constraints $c_1 \dots c_m$. A constraint $c_m(v_{m1} \dots v_{ml})$ is a predicate which is defined on the Cartesian product $D_{m1} \times \dots \times D_{ml}$. Solving a CSP means satisfying all constraints by an instantiation of the variables.

Schiex et al. [12] extend the CSP to a valued CSP (VCSP), which is usually called a Constraint Optimization Problem (COP) in literature such as [4,7,17]. We will use the term COP in this paper. In a COP, each constraint has its own associated utility function, and there is a global utility function that aggregates the values of utilities over all the constraints. A COP is solved by maximizing the global utility function.

In a Distributed Constraint Satisfaction Problem (DCSP) [16], the variables of a CSP are distributed among agents. Analogously, the variables of a COP are distributed among agents in a Distributed Constraint Optimization Problem (DCOP) [2]. In this paper, without loss of generality, each agent i will control exactly one variable v_i.

Chapman et al. [2] divide DCOP algorithms into: distributed complete, local iterative message-passing and local iterative approximate best-response algorithms. This paper will focus on algorithms from the last category.

Local iterative approximate best-response algorithms exchange only their state with their neighbors [2]. All algorithms discussed in this paper are of this class. Chapman et al. [2] decompose such algorithms into three components: (1) **the state evaluation**, where a target function is used to evaluate possible states, (2) **the decision rule**, which defines what action to take given the results of the state evaluation, and (3) **the adjustment schedule** that determines when each agent updates its state.

2.2 Distributed Stochastic Algorithm

The Distributed Stochastic Algorithm (DSA) is used as the baseline algorithm in this paper. Zhang et al. [17] have compared different variations of DSA, of which

DSA-B has the best performance. They did not define which state to choose if there are multiple maximizing states. Therefore, we will use the definition of DSA-$\overline{B}$ by Arshad and Silaghi [1], which states that a maximizing state shall be chosen randomly if multiple exist. If not stated otherwise, the term DSA refers to DSA-$\overline{B}$ in this paper.

DSA, as shown in Algorithm 1, chooses a candidate state *candidateState* that maximizes the target function U_i. The target function calculates the utility of a variable given the states of itself and its neighbors s_{-i} at time $t-1$. The candidate state is chosen with probability p only if it either (1) has the same utility as the current state (*currentUtility*) and does not satisfy all constraints, or (2) if it improves the utility of the current state.

Algorithm 1 DSA (agent i, step t, degree of parallelism p)

```
$candidateState := \arg\max_{\overline{s} \in D_i} \mathrm{U}_i(\overline{s}, s_{-i}(t-1))$
$currentUtility := \mathrm{U}_i(s_i(t-1), s_{-i}(t-1))$
$\Delta := \mathrm{U}_i(candidateState, s_{-i}(t-1)) - currentUtility$
if random$(0,1) < p$ then
   if $\Delta > 0 \vee (\Delta = 0 \wedge$ not allConstraintsSatisfied$)$ then
      return $candidateState$
   end if
end if
return $s_i(t-1)$
```

2.3 Distributed Simulated Annealing

Distributed Simulated Annealing (DSAN) [1] avoids the greedy behavior of DSA. DSAN has a decision rule that simulates physical annealing and is shown in Algorithm 2. It chooses a candidate state randomly, which is accepted if it improves the utility of the current state, or with probability p_t otherwise. $p_t = e^{\Delta t^k / const}$ represents a decreasing schedule of temperatures. We used the values $k = 2$ and $const = 1000$ in our experiments, as done in [1].

Verman et al. [14] introduced a further parameter γ, which fixes oscillation when solving under-constrained problems. It is used as p_t if the utility of the candidate state is equal to the utility of the current state. We used $\gamma = 0.001$ in our experiments, as done in [14].

Chapman et al. [2] suggested the same adjustment schedule for DSAN as for DSA. Therefore, Algorithm 2 also includes the degree of parallel executions p.

2.4 Ranked-Distributed Stochastic Algorithm

Verman et al. [13] introduced the Ranked Distributed Stochastic Algorithm (RDSA), a variation of DSA that uses a modified PageRank [10] algorithm. They found that DSA converges quickly to large areas without conflicts, but

Algorithm 2 DSAN (agent i, step t, degree of parallelism p, k, $const$, γ)

```
$candidateState := \text{randomElement}(D_i)$
$currentUtility := \text{U}_i(s_i(t-1), s_{-i}(t-1))$
$\Delta := \text{U}_i(candidateState, s_{-i}(t-1)) - currentUtility$
$p_t := e^{\Delta t^k / const}$
**if** $\Delta = 0$ **then**
  $p_t := \gamma$
**end if**
**if** $\text{random}(0,1) < p$ **then**
  **if** $\Delta > 0 \vee \text{random}(0,1) < p_t$ **then**
    **return** $candidateState$
  **end if**
**end if**
**return** $s_i(t-1)$
```

with conflicts at the borders. Therefore, they proposed to assign weights to the agents that correlate with the number and connectivity of allying agents and help important areas to influence less important areas.

Using those weights, a rank is computed for each agent. Unlike with the original PageRank algorithm, only allying agents are included in the calculation of the ranks. Allying agents are neighbors that are not in conflict with the agent. This definition is specific to graph coloring problems and the next section shows a generalization of it, including an accordant implementation of RDSA.

RDSA performs better than DSA during a few iteration steps in the beginning, but it converges quickly to a local optimum of lower quality than DSA. Verman et al. [13] explained the reason for this behavior on an 8x8 grid like the one shown in Fig. 1. The two heat maps on the right show that the vertices in positions (5, 5) and (5, 6) are in conflict, but they both have high ranks and are supported by the same area.

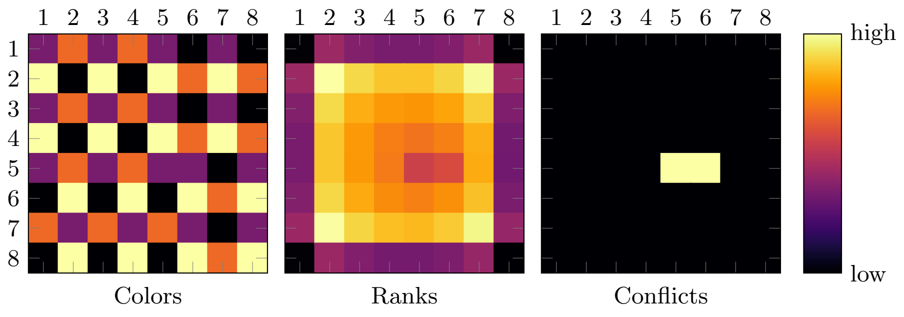

Fig. 1. Coloring problem for an 8x8 grid [13]

3 Generalizing Graph Coloring Problems

DSA bases its decision rule on the number of conflicts, i.e. on the number of unsatisfied constraints. In addition, the ranked algorithm in [13] partitions the neighbors of a vertex into allying and opposing vertices, depending on the satisfaction of the constraints. The definition of a satisfied constraint is obvious for graph coloring problems. However, the result of evaluating a constraint in a DCOP is not limited to two values in general.

Satisfying all constraints of a graph coloring problem is a sufficient but not necessary condition of an optimal solution. This is in contrast to a Nash equilibrium, which is a necessary but not sufficient condition of an optimal solution. In accordance with graph coloring problems, we define a constraint of a general DCOP as satisfied if it yields its optimal utility. That is equivalent for graph coloring problems with more than one color.

Equation (1) shows the condition for the satisfaction of all constraints of agent i. U_i is the utility of agent i, $s_i(t-1)$ is the agent's state at time $t-1$, $s_{-i}(t-1)$ are the neighbors' states at time $t-1$, D_i are the possible states of agent i, and D_{-i} are the possible states of the neighbors.

$$\mathrm{U}_i(s_i(t-1), s_{-i}(t-1)) = \max_{\substack{\overline{s_i} \in D_i \\ \overline{s_{-i}} \in D_{-i}}} \mathrm{U}_i(\overline{s_i}, \overline{s_{-i}}) \tag{1}$$

3.1 Generalized RDSA

Algorithm 3 RDSA (agent i, step t, degree of parallelism p, *baseRank*, damping factor d)

```
$rank := baseRank + d * \mathrm{rankSum}(s_i(t-1), s_{-i}(t-1))$
$candidateState := \arg\max_{\overline{s} \in D_i} \mathrm{target}(\overline{s}, s_{-i}(t-1))$
$currentExpectedUtility := \mathrm{target}(s_i(t-1), s_{-i}(t-1))$
$\Delta := \mathrm{target}(candidateState, s_{-i}(t-1)) - currentExpectedUtility$
if $\mathrm{random}(0,1) < p$ then
  if $\Delta > 0 \vee (\Delta = 0 \wedge$ not allConstraintsSatisfied$)$ then
    return $candidateState, \mathrm{sendRank}(rank)$
  end if
end if
return $s_i(t-1), \mathrm{sendRank}(rank)$
```

Algorithm 3 shows our implementation of RDSA, which is a generalization of the original RDSA. RDSA uses a different target function than DSA for the state evaluation. Equation (4) shows the target function for agent i and state $\overline{s_i}$. $\mathrm{U}_{i,j}$ is the utility of the constraint between agent i and j, D_i are the possible states of agent i, $-i$ are the neighbors of agent i, and $rank_j$ is the rank received from neighbor j.

$$\mathrm{U}_{\max}(i,j) = \max_{\overline{s_i} \in D_i} \max_{\overline{s_j} \in D_j} \mathrm{U}_{i,j}(\overline{s_i}, \overline{s_j}) \tag{2}$$

$$\mathrm{U}_{\min}(i,j) = \min_{\overline{s_i} \in D_i} \min_{\overline{s_j} \in D_j} \mathrm{U}_{i,j}(\overline{s_i}, \overline{s_j}) \tag{3}$$

$$\mathrm{target}_i(\overline{s_i}, s_{-i}) = \sum_{\overline{j} \in -i} rank_{\overline{j}} \frac{2\,\mathrm{U}_{i,\overline{j}}(\overline{s_i}, s_{\overline{j}}) - \mathrm{U}_{\max}(i,\overline{j}) - \mathrm{U}_{\min}(i,\overline{j})}{\mathrm{U}_{\max}(i,\overline{j}) - \mathrm{U}_{\min}(i,\overline{j})} \tag{4}$$

We assume that each constraint has at least two states with distinct utilities, and thus that the denominator of (4) is always greater than zero. Constraints that have only a single possible utility can be eliminated, transforming the problem into an equivalent problem.

There is a difference between the algorithm shown in [13] and Algorithm 3 in the way how the candidate state is chosen. Algorithm 3 might choose the current state if it is a maximizing state, while the other one never chooses the current state as the candidate state. However, Verman et al. [13] used for their experiments an implementation of RDSA that chooses the candidate state in a way compatible with our implementation.

The calculation of the rank uses the function rankSum as defined in (5), which sums up the ranks of the neighbors, weighted by their utilities. We used the values $baseRank = 0.15$ and $d = 0.85$ in our experiments, as done in [13].

$$\mathrm{rankSum}_i(s_i, s_{-i}) = \sum_{\overline{j} \in -i} rank_{\overline{j}} \frac{\mathrm{U}_{i,\overline{j}}(s_i, s_{\overline{j}}) - \mathrm{U}_{\min}(i,\overline{j})}{\mathrm{U}_{\max}(i,\overline{j}) - \mathrm{U}_{\min}(i,\overline{j})} \tag{5}$$

Unlike [13], our implementation calculates the rank before changing the state. Our preliminary experiments showed that this performed better than the original RDSA. A possible explanation for this is that sending the old rank to the neighbors could avoid some oscillating behavior.

The agent not only sends its state to the neighbors, but also its rank split among the neighbors. This is done by the function sendRank, which calculates for each neighbor j of agent i the rank to be sent using (7).

$$\mathrm{U}_{\min}(i) = \min_{\overline{j} \in -i} \mathrm{U}_{\min}(i,\overline{j}) \tag{6}$$

$$\mathrm{sendRank}_{i,j}(rank) = rank \frac{\mathrm{U}_{\max}(i,j) - \mathrm{U}_{\min}(i)}{\sum_{\overline{j} \in -i} (\mathrm{U}_{\max}(i,\overline{j}) - \mathrm{U}_{\min}(i))} \tag{7}$$

Even with the improvements determined by calculating the rank before changing the state, the experiments show that the solution quality is still lower than for the original DSA.

4 New Ranked Algorithms

The previous section showed a problem of RDSA. We have, thus, looked for a way to overcome it. The main idea is to create a gradient that gains enough momentum to escape a local optimum. We introduce the Modified Ranked Distributed Stochastic Algorithm (MRDSA), where we modify the own rank after the next move has been computed, but before the new rank is communicated to the neighbors.

4.1 MRDSA

Based on Algorithm 4, we tested different linear functions (affine transformations) on the rank, that are either applied when the own state changes, or when it stays the same. In particular, the own rank shall (1) be increased if the own state changed, and/or (2) the rank shall be decreased if the state did not change. That way, vertices that did not change their state recently shall be activated to change their state, avoiding a fall-back to the local optimum.

Algorithm 4 MRDSA (agent i, step t, degree of parallelism p, *baseRank*, damping factor d, a_1, b_1, a_2, b_2)

```
$rank := baseRank + d * \text{rankSum}(s_i(t-1), s_{-i}(t-1))$
$candidateState := \arg\max_{\overline{s} \in D_i} \text{target}(\overline{s}, s_{-i}(t-1))$
$currentExpectedUtility := \text{target}(s_i(t-1), s_{-i}(t-1))$
$\Delta := \text{target}(candidateState, s_{-i}(t-1)) - currentExpectedUtility$
if $\text{random}(0,1) < p$ then
  if $\Delta > 0 \lor (\Delta = 0 \land candidateState \neq s_i(t-1) \land$ not $\text{allConstraintsSatisfied})$ then
    $rank := rank * a_1 + b_1$
    return $candidateState, \text{sendRank}(rank)$
  else
    $rank := rank * a_2 + b_2$
  end if
end if
return $s_i(t-1), \text{sendRank}(rank)$
```

We tried to minimize the number of conflicts of graph coloring problems at the 40th iteration step, since Verman et al. [13] highlighted the first 40 steps in their results. The best configuration in these preliminary experiments, which we used in our further experiments, has the values $a_1 = 1$, $b_1 = 0$, $a_2 = 0.75$ and $b_2 = -0.075$ in Algorithm 4. That means the rank is not changed if the state changed, but it is decreased if the state did not change. While it improves the performance of RDSA, it does not outperform DSA at step 40. However, after more iteration steps, it performs better than DSA in the long term.

4.2 MRDSAN and RDSAN

We have extended DSAN according to MRDSA. We will call that algorithm Modified Ranked Distributed Simulated Annealing (MRDSAN). Algorithm 5 shows how MRDSAN works. As for DSAN and MRDSA, we used the values

$k = 2$, $const = 1000$, $\gamma = 0.001$, $baseRank = 0.15$, $d = 0.85$, $a_1 = 1$, $b_1 = 0$, $a_2 = 0.75$ and $b_2 = -0.075$ in our experiments.

Algorithm 5 MRDSAN (agent i, step t, degree of parallelism p, k, $const$, γ, $baseRank$, damping factor d, a_1, b_1, a_2, b_2)

```
$rank := baseRank + d * \text{rankSum}(s_i(t-1), s_{-i}(t-1))$
$candidateState := \text{randomElement}(D_i)$
$currentExpectedUtility := \text{target}(s_i(t-1), s_{-i}(t-1))$
$\Delta := \text{target}(candidateState, s_{-i}(t-1)) - currentExpectedUtility$
$p_t := e^{\Delta t^k / const}$
if $\Delta = 0$ then
  $p_t := \gamma$
end if
if $\text{random}(0,1) < p$ then
  if $\Delta > 0 \vee (candidateState \neq s_i(t-1) \wedge \text{random}(0,1) < p_t)$ then
    $rank := rank * a_1 + b_1$
    return $candidateState, \text{sendRank}(rank)$
  else
    $rank := rank * a_2 + b_2$
  end if
end if
return $s_i(t-1), \text{sendRank}(rank)$
```

As a counterpart of RDSA, we will use the term Ranked Distributed Simulated Annealing (RDSAN) for a special case of MRDSAN with the values $a_1 = 1$, $b_1 = 0$, $a_2 = 1$ and $b_2 = 0$ in Algorithm 5.

5 Experiments

After introducing the algorithms used in the experiments, and a generalization for randomized DCOPs, this section presents the data sets used in the experiments and how the solution quality is measured.

5.1 Data Sets

We used graph coloring and randomized DCOP data sets from [15] for the evaluation of the algorithms. In addition, we generated large scale problems. Table 1 shows an overview of the data sets. Link density means the average number of neighbors per vertex. The graph coloring problems with 40 vertices were used in [6,9,13]. The randomized DCOP data sets were used in [6].

Our goal was to compare the algorithms against the optimal solutions, so we solved the problems with complete algorithms. We used different algorithms (Adopt [9], DPOP [11] and SynchBB [5]) because the problems were so complex that we could not solve them with a single algorithm. We were not able to compute

Table 1. Data sets used in the experiments

Type	Vertices	Link density	Colors	Problems	Over-constrained problems	Runs
Coloring	40	2	3	25	9	1,000
Coloring	40	3	3	25	25	1,000
Coloring	100,000	2	3	25	0	1
Coloring	100,000	3	3	25	0	1
Random	30	2	—	25	—	1,000
Random	30	3	—	25	—	1,000
Random	40	2	—	25	—	1,000

the optimal solutions for the randomized DCOP data set with 40 vertices and link density 3, so we excluded that data set. In order to avoid such issues with the large scale problems, the problems were generated in a way such they are not over-constrained, similar to the method in [3].

As presented in Table 1, we ran the algorithms synchronously for 1,000 iteration steps. Each configuration was run 1,000 times on the small problems ($\leq$ 40 vertices), and once on the large scale problems, since they took much more time to be solved.

Unlike [13], we initialized the variables of each problem randomly at each run. We found that this performs better, in particular with DSA, which is used as a baseline in our experiments. Furthermore, besides graph coloring problems, we also evaluated randomized DCOPs. Uniformly initializing those problems would make it more difficult to statistically compare the performance of different problems during the first iteration steps.

We also found in preliminary experiments that a degree of parallel executions of 0.6 performs best in general, in particular with DSA. Therefore, and to be consistent, we used a degree of parallel executions of 0.6 for all experiments.

For statistical tests, we calculated the variance of each small problem separately. For the large scale problems, we calculated the variance by link density. Unless stated otherwise, all statistically significant results are based on two-tailed Welch's t-tests with a confidence level of 99.9%.

5.2 Solution Quality

The quality of a solution for a graph coloring problem can be measured by the number of conflicting constraints, as done in [13]. For randomized DCOPs, the global utility can be used as a solution quality indicator. However, that makes it difficult to compare the solution quality of different problems.

Therefore, we have normalized the solution quality as shown in (11). Equation (8) calculates the expected utility at the initial state with random initialization, (9) the optimal utility as computed with a complete algorithm, and (10) the global utility at time t. $\mathrm{U}_{i,j}$ is the utility between agent i and j, returning 0 if i

and j are not neighbors. U_i is the utility between agent i and its neighbors, $s_i(t)$ is the agent's state at time t, $s_{-i}(t)$ are the neighbors' states at time t, and D_i are the possible states of agent i.

All utilities are divided by two because each constraint is counted twice. However, that does not influence the normalized solution quality. With that measure, an algorithm is expected to start with a solution quality of 0 and to approach a solution quality of 1 over time. A complete algorithm always terminates with a solution quality of 1.

$$\mathit{expectedUtility} = \frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\frac{1}{|D_i|\,|D_j|}\sum_{\substack{\overline{s_i}\in D_i\\ \overline{s_j}\in D_j}}\mathrm{U}_{i,j}(\overline{s_i},\overline{s_j}) \tag{8}$$

$$\mathit{maxUtility} = \max_{\substack{(\overline{s_1},\ldots,\overline{s_n})\in\\ D_1\times\ldots\times D_n}}\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\mathrm{U}_{i,j}(\overline{s_i},\overline{s_j}) \tag{9}$$

$$\mathrm{U}(t) = \frac{1}{2}\sum_{i=1}^{n}\mathrm{U}_i(s_i(t), s_{-i}(t)) \tag{10}$$

$$\mathrm{solutionQuality}(t) = \frac{\mathrm{U}(t) - \mathit{expectedUtility}}{\mathit{maxUtility} - \mathit{expectedUtility}} \tag{11}$$

6 Results

After defining the experiments in the previous section, this section presents their results, first for the graph coloring problems, then for the randomized DCOPs.

The newly introduced algorithms MRDSA and MRDSAN perform well with the graph coloring data sets. On these data sets, MRDSA is superior to all other tested algorithms in the medium term, and MRDSAN in general outperforms MRDSA and the other algorithms in the longer term. For randomized DCOP data sets, RDSAN is superior to all other tested algorithms in the longer term.

6.1 Graph Coloring

40 Vertices. Figure 2 compares the two algorithms that have the best performance in the longer term to DSA and DSAN, split by link density. MRDSA is superior during the steps 191 to 534 with a link density of 2, and during the steps 261 to 824 with a link density of 3. MRDSAN is superior during the steps 603 to 1,000 with a link density of 2. However, with a link density of 3, it never outperforms MRDSA with at least 99% confidence. At step 1,000, MRDSAN performs better than MRDSA with a confidence level of 87%.

100,000 Vertices. Figure 3 shows the solution quality for the graph coloring problems with 100,000 vertices. The results look similar to the graph coloring

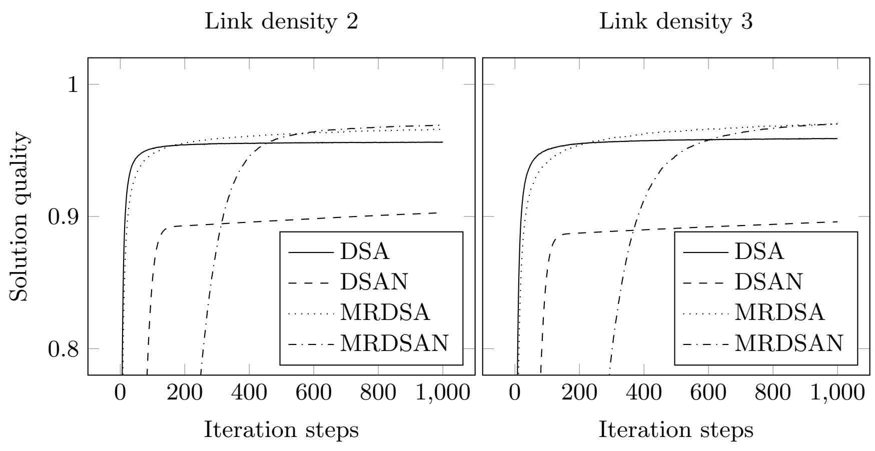

Fig. 2. Graph coloring with 40 vertices

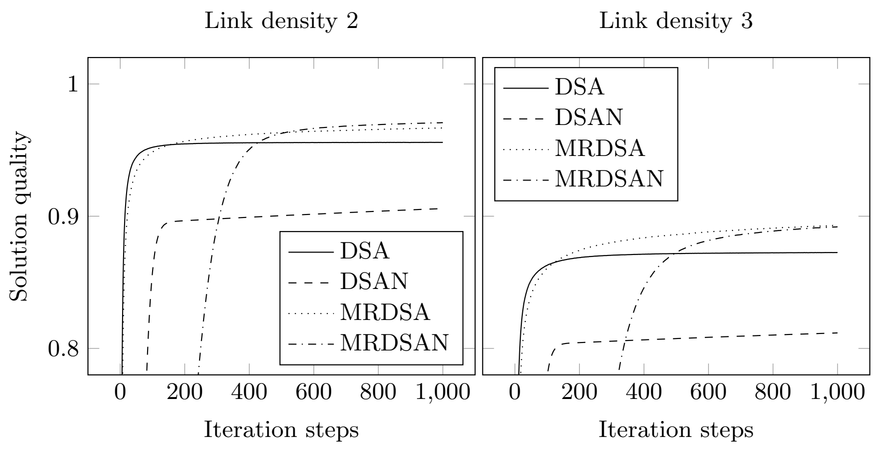

Fig. 3. Graph coloring with 100,000 vertices

problems with 40 vertices. One difference is that, with a link density of 3, MRDSA outperforms MRDSAN during all 1,000 steps.

However, probably more notable is that all tested algorithms perform worse with the problems with link density 3. We see two possible reasons for this: (1) The higher number of vertices makes it more difficult to solve the problems, and/or (2) it is more difficult to approach an optimal solution because the problems are not over-constrained, i.e. an optimal solution must not have any conflicts.

6.2 Randomized DCOPs

30 Vertices. Figure 4 shows the solution quality for the randomized DCOPs with 30 vertices. It compares the algorithms DSA, DSAN and MRDSA to the algorithm RDSAN, which outperformed all other tested algorithms in the longer term. RDSAN is superior during the steps 329 to 1,000 with a link density of 2, and during the steps 358 to 1,000 with a link density of 3. MRDSA outperforms DSA during the steps 66 to 1,000 with a link density of 2.

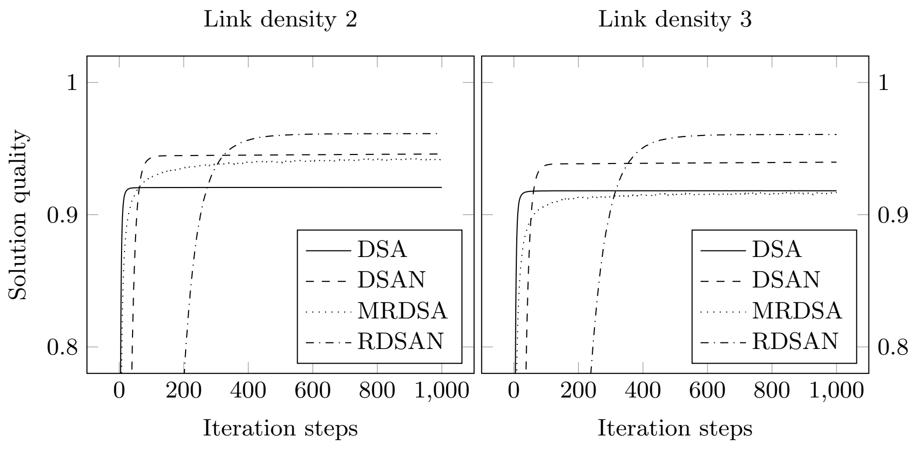

Fig. 4. Randomized DCOPs with 30 vertices

40 Vertices. Figure 5 shows the solution quality for the randomized DCOPs with 40 vertices and a link density of 2. Similar to the results with 30 vertices, RDSAN is superior during the steps 318 to 1,000, and MRDSA outperforms DSA during the steps 68 to 1,000.

7 Limitations and Future Work

MRDSA, MRDSAN and RDSAN showed promising results. However, we think that these new algorithms, especially MRDSAN, have not realized their maximum

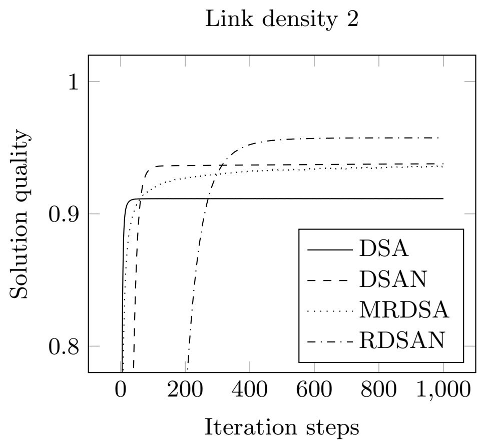

Fig. 5. Randomized DCOPs with 40 vertices

potential. MRDSAN has 10 configuration parameters as shown in Algorithm 5. While we tried various combinations of these parameters in preliminary experiments, we could not test all reasonable combinations due to time reasons. Therefore, it would make sense to explore more settings.

Furthermore, while the new algorithms were designed based on intuition, the actual effect was only determined empirically. Having a better theoretical understanding of the interactions between the various configuration parameters could help to further develop the algorithms.

Another approach would be to develop ranked algorithms that are based on other algorithms than DSA or DSAN, for example on Joint Strategy Fictitious Play with Inertia [8], which, unlike DSA or DSAN, takes into account previous neighbor actions. In addition, strategies that switch between different algorithms could also be a way to improve the performance, as discussed in [13].

8 Conclusions

This paper has introduced the algorithms MRDSA, MRDSAN and RDSAN, which enhance RDSA of [13] and are based on DSA and DSAN respectively. MRDSA and MRDSAN outperformed all other tested algorithms in the longer term on graph coloring problems. MRDSA converged faster than MRDSAN in the beginning, but MRDSAN generally performed better in the long term. With randomized DCOPs, RDSAN surpassed all other tested algorithms in the longer term.

Acknowledgments. We would like to thank the Hasler Foundation for supporting the Signal/Collect Project, which we used to implement our algorithms, under grant number 11072.

References

1. Arshad, M., Silaghi, M.: Distributed Simulated Annealing and Comparison to DSA. In: Proceedings of the fourth International Workshop on Distributed Constraint Reasoning (DCR-03) (2003)
2. Chapman, A., Rogers, A., Jennings, N., Leslie, D.: A Unifying Framework for Iterative Approximate Best-response Algorithms for Distributed Constraint Optimization Problems. Knowledge Engineering Review 26(4), 411–444 (2011)
3. Chapman, A., Rogers, A., Jennings, N.: Benchmarking Hybrid Algorithms for Distributed Constraint Optimisation Games. Autonomous Agents and Multi-Agent Systems 22, 385–414 (2011)
4. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. In: AAMAS '08: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 639–646. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2008)
5. Hirayama, K., Yokoo, M.: Distributed Partial Constraint Satisfaction Problem. In: Principles and Practice of Constraint Programming. pp. 222–236 (1997)
6. Maheswaran, R., Pearce, J., Tambe, M.: A Family of Graphical-Game-Based Algorithms for Distributed Constraint Optimization Problems. Coordination of Large-Scale Multiagent Systems pp. 127–146 (2006)
7. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 310–317. IEEE Computer Society (2004)
8. Marden, J.R., Arslan, G., Shamma, J.S.: Joint Strategy Fictitious Play With Inertia for Potential Games. IEEE Transactions on Automatic Control 54(2), 208–220 (2009)
9. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: An Asynchronous Complete Method for Distributed Constraint Optimization. In: AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 161–168. ACM, New York, NY, USA (2003)
10. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web (1999)
11. Petcu, A., Faltings, B.: A Scalable Method for Multiagent Constraint Optimization. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 266–271. Professional Book Center, Denver, Colo. (2005)
12. Schiex, T., Fargier, H., Verfaillie, G., et al.: Valued constraint satisfaction problems: Hard and easy problems. In: International Joint Conference on Artificial Intelligence. vol. 14, pp. 631–639. Citeseer (1995)
13. Verman, M., Stutz, P., Bernstein, A.: Solving Distributed Constraint Optimization Problems Using Ranks. In: Statistical Relational AI. Papers Presented at the Twenty-Eighth AAAI Conference on Artificial Intelligence. pp. 125–130. AAAI Press, Palo Alto, California (2014)
14. Verman, M., Stutz, P., Hafen, R., Bernstein, A.: Solving Big Distributed Constraint Optimization Problems. In: Autonomous Agents and Multi-Agent Systems at Scale Workshop, International Workshop on Massive Multi-Agents, in conjunction with AAMAS 2015. Istanbul, Turkey (2015)
15. Yin, Z.: USC DCOP Repository (2008), `http://teamcore.usc.edu/dcop`

16. Yokoo, M., Ishida, T., Durfee, E., Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In: Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on. pp. 614–621. IEEE (1992)
17. Zhang, W., Wang, G., Wittenburg, L.: Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance. In: Proceedings of AAAI Workshop on Probabilistic Approaches in Search (2002)