Anytime Exploration for Distributed Constraint Optimization

Hilla Peled and Roie Zivan
Industrial Engineering and Management department,
Ben Gurion University,
Beer-Sheva, Israel
{*hillapel,zivanr*}*@bgu.ac.il*

Abstract. Distributed Constraint Optimization Problems (DCOPs) are an elegant model for representing and solving many realistic combinatorial problems which are distributed by nature. DCOPs are NP-hard and therefore most recent studies consider incomplete (local) search algorithms for solving them. Distributed local search algorithms can be used for solving DCOPs. However, because of the differences between the global evaluation of a system's state and the private evaluation of states by agents, agents are unaware of the global best state which is explored by the algorithm. Previous attempts to use local search algorithms for solving DCOPs reported the state held by the system at the termination of the algorithm, which was not necessarily the (global) best state explored.
A general framework for implementing distributed local search algorithms for DCOPs was proposed in [24]. The framework makes use of a BFS-tree in order to accumulate the costs of the system's state in its different steps and to propagate the detection of a new best step when it is found. The resulting framework enhances local search algorithms for DCOPs with the *anytime* property. However, since most local search algorithms are mostly exploitive the potential of the anytime framework has not been explored.
We propose a set of increased exploration heuristics that exploit the proposed anytime framework. Our empirical study reveals the advantage of the use of the proposed heuristics in the anytime framework over state of the art local search algorithms.

1 Introduction

The Distributed Constraint Optimization Problem (*DCOP*) is a general model for distributed problem solving that has a wide range of applications in Multi-Agent Systems and has generated significant interest from researchers [1, 8, 11, 13, 15, 22]. DCOPs are composed of agents, each holding one or more variables. Each variable has a domain of possible value assignments. Constraints among variables (possibly held by different agents) assign costs to combinations of value assignments. Agents assign values to their variables and communicate with each other, attempting to generate a solution that is globally optimal with respect to the costs of the constraints [11, 12].

There is a wide scope in the motivation for research on DCOPs, since they can be used to model many every day combinatorial problems that are distributed by nature.

Some examples are the *Nurse Shifts assignment problem* [16, 6], the *Sensor Network Tracking problem* [22], and *Log Based Reconciliation problem* [2].

DCOPs represent real life problems that cannot or should not be solved centrally for several reasons, among them lack of autonomy, single point of failure, and privacy of agents.

A number of studies on DCOPs presented complete algorithms [11, 13, 4]. However, since DCOPs are NP-hard, there is a growing interest in the last few years in local (incomplete) DCOP algorithms [7, 22, 24, 17, 19]. Although local search does not guarantee that the obtained solution is optimal, it is applicable for large problems and compatible with real time applications.

The general design of most state of the art local search algorithms for DCOPs is synchronous (DALO is the only published exception). In each step of the algorithm an agent sends its assignment to all its neighbors in the constraint network and receives the assignment of all its neighbors. They differ in the method agents use to decide whether to replace their current value assignments to their variables, e.g., in the max gain messages algorithm (MGM) [7], the agent that can improve its state the most in its neighborhood replaces its assignment. A stochastic decision whether to replace an assignment is made by agents in the distributed stochastic algorithm (DSA) [22].

In the case of centralized optimization problems, local search techniques are used when the problems are too large to perform a complete search. Traditionally, local search algorithms maintain a complete assignment for the problem and use a goal function in order to evaluate this assignment. Different methods which balance between exploration and exploitation are used to improve the current assignment of the algorithm [5, 14, 18]. An important feature of most local search algorithms is that they hold the best assignment that was found throughout the search. This makes them *anytime* algorithms, i.e., the quality of the solution can only increase or remain the same if more iterations of the algorithm are performed [23]. This feature cannot be applied in a distributed environment where agents are only aware of the cost of their own assignment (and maybe their neighbors too) but no one actually knows when a good global solution is obtained.

In [24] a general framework for enhancing local search algorithms for DCOPs which follows the general synchronous structure with the *anytime* property, was proposed. In the anytime local search framework for DCOPs (ALS_DCOP), the quality of each state is accumulated via a *Breadth First Search tree* (BFS-tree) structure. Agents receive the information about the quality of the recent states of the algorithm from their $children$ in the BFS-tree, calculate the resulting quality including their own contribution according to the goal function, and pass it to their parents. The *root agent* makes the final calculation of the cost for each state and propagates down the tree the index number of the most successful state. When the search is terminated, all agents hold the assignment of the best state according to the global goal function.

In order to produce the best state out of m steps, the algorithm must run $m + (2 * h)$ synchronous steps where h is the height of the tree used. ALS_DCOP does not require agents to send any messages beside the messages sent by the original algorithm. The space requirements for each agent are $O(h)$ and it preserves a high level of privacy (see [24] for details).

In this paper we study the potential of the proposed framework by proposing a set of exploration methods (heuristics) which exploit the anytime property by introducing extreme exploration to exploitive algorithms. We present an intensive empirical evaluation of the proposed methods on three different benchmarks for DCOPs. The proposed methods find solution of higher quality than state of the art algorithms when implemented within the anytime local search framework.

The rest of the paper is organized as follows: Section 2 describes the distributed constraint optimization problem ($DCOP$). State of the art local search algorithms for solving $DCOPs$, will be presented in Section 3. Section 4 presents ALS_DCOP. Section 5 presents a set of heuristics which increase the exploration of standard local search algorithms. We evaluate the performance of the proposed heuristics in Section 6 Section 7 presents our conclusions.

2 Distributed Constraint Optimization

A $DCOP$ is a tuple $< \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} >$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, ..., A_n$. $\mathcal{X}$ is a finite set of variables X_1,X_2,...,X_m. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains D_1, D_2,...,D_m. Each domain D_i contains the finite set of values which can be assigned to variable X_i. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary DCOP* is a DCOP in which all constraints are binary. An *assignment* (or a label) is a pair including a variable, and a value from that variable's domain. A *partial assignment* (PA) is a set of assignments, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists a \in D_i \wedge (X_i, a) \in PA\}$. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if $X_{i_1}, X_{i_2}, \ldots, X_{i_k} \in vars(PA)$. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *full assignment* is a partial assignment that includes all the variables ($vars(PA) = \mathcal{X}$). A *solution* is a full assignment of minimal cost.

In this paper, we will assume each agent owns a single variable, and use the term "agent" and "variable" interchangeably. We will assume that constraints are at most binary and the delay in delivering a message is finite [11, 21]. Agents are aware only of their own topology (i.e. only of their own neighbors in the constraints network and the constraints that they personally and privately hold).

3 Local Search for Distributed Constraints problems

The general design of most state of the art local search algorithms for Distributed Constraint *Satisfaction and Optimization* Problems ($DisCSPs$ and $DCOPs$) is synchronous. In each step of the algorithm an agent sends its assignment to all its neighbors in the constraint network and receives the assignment of all its neighbors. We present as an example an algorithm that applies to this general framework, the *Distributed Stochastic Algorithm* (DSA) [22]. The algorithms is presented following the

recent version of [22]. Notice that these algorithms were first designed for distributed *satisfaction* problems in which a solution must not violate any constraint, but they can be applied as is to *Distributed Max CSPs* (where the optimal solution is the complete assignment with the smallest number of violated constraints) which is a special type of DCOPs. Thus in our description we consider an improvement a decrease in the number of violated constraints (as in Max-CSPs).

The basic idea of the DSA algorithm is simple. After an initial step in which agents pick some value for their variable (random according to [22]), agents perform a sequence of steps until some termination condition is met. In each step, an agent sends its value assignment to its neighbors in the constraints graph and receives the assignments of its neighbors.[1] After collecting the assignments of all its neighbors, an agent decides whether to keep its value assignment or to change it. This decision which is made stochastically has a large effect on the performance of the algorithm. According to [22], if an agent in DSA cannot improve its current state by replacing its current value, it does not replace it. If it can improve, it decides whether to replace the value using a stochastic strategy (see [22] for details on the possible strategies and the difference in the resulting performance). A sketch of DSA is presented in Figure 1. After a random value is assigned to the agent's variable (line 1) the agent performs a loop (each iteration of the loop is a step of the algorithm) until the termination condition is met. In each step the agent sends its value assignment to all its neighbors and collects the assignments of all its neighbors (lines 3,4). According to the information it receives, it decides whether to replace its assignment; when the decision is positive it assigns a new value to its variable (lines 5,6).

DSA

```
1. value ← ChooseRandomValue()
2. while (no termination condition is met)
3.     send value to neighbors
4.     collect neighbors' values
5.     if (ReplacementDecision())
6.         select and assign the next value
```

Fig. 1. Standard DSA for DisCSPs.

An example of a DisCSP is presented in Figure 2. Each of the agents has a single variable with the values a and b in its domain. Dashed lines connect constrained agents and all constraints are equality constraints. Although DSA is a uniform algorithm, i.e., the algorithm does not assume the existence of agents' identifiers, we added identifiers to the figure to make it easier to describe.

Before the first step of the algorithm each agent selects a random number. Assume agents 1, 3, and 5 selected a and agents 2 and 4 selected b. In the first step all agents can improve their states by changing their assignment. Following a stochastic decision

[1] In this paper we follow the general definition of a $DCOP$ and a $DisCSP$ which does not include a synchronization mechanism. If such a mechanism exists, agents in DSA can send value messages only in steps in which they change their assignments.

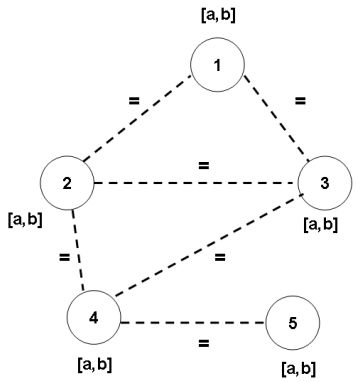

Fig. 2. An example of a DisCSP.

only agents 2 and 5 replace their assignment. Now agents 1, 2, and 3 hold a and agents 4 and 5 hold b. At this step only agent 4 can replace its assignment and in the next only agent 5 can replace. In the resulting state, all agents are holding a and the algorithm is terminated.

4 Anytime Local Search framework for DCOPs

The ALS_DCOP framework enhances $DCOP$ synchronous local search algorithms with the *anytime* property [24]. In the proposed framework, ALS_DCOP, a tree is used as in $ADOPT$ [11] and $DPOP$ [13]. In contrast to $ADOPT$ and $DPOP$ that require the use of a pseudo-tree, the only requirement in ALS_DCOP is that every agent has a parent route to the root agent. Thus, a *Breadth First Search* (BFS) tree on the constraint graph can be used. The BFS-tree structure is used in order to accumulate the cost of agents assignments in the different states during the execution of the algorithm. Each agent calculates the cost of the sub-tree it is a root of in the BFS-tree and passes it to its parents. The root agent calculates the complete cost of each state and if it is found to be the best state found so far, propagates its index to the rest of the agents. Each agent A_i is required to hold its assignments in the last $2 * d_i$ steps where d_i is the length of the route of parents in the BFS-tree from A_i to the root agent and is bounded by the height of the BFS-tree (h).

Next, we describe in details the actions agents perform in the ALS_DCOP framework regardless of the algorithm in use. In each step of the algorithm an agent collects from its children in the BFS-tree the calculation of the cost of the sub-tree of which they are the root of. When it receives the costs for a step j from all its children, it adds its own cost for the state in step j and sends the result to its parent. When the root agent receives the calculations of the cost of step j from all its children, it calculates the global state cost. If it is better than the best state found so far, in the next step it will inform all its children that the state in step j is the best state found so far. Agents which are informed of the new best step store their assignment in that step as the best assignment and pass the information about the best index to their children in the next

DSA in ALS_DCOP

1. $height \leftarrow$ height in the BFS- tree
2. $dist \leftarrow$ distance from root
3. $best \leftarrow null$
4. $best\_index \leftarrow null$
5. $current\_step \leftarrow 0$
6. **if** (root)
7. $\quad best\_cost \leftarrow \infty$
8. $value\_current \leftarrow$ ChooseRandomValue()
9. **while** $(current\_step < (m + dist + height))$
10. $\quad$ send value and $cost\_i$ to parent
11. $\quad$ send value to non tree neighbors
12. $\quad$ send value and $best\_index$ to children
13. $\quad$ collect neighbors' values
14. $\quad cost\_i \leftarrow$ CalculateStepCost$(current_step - height)$
15. $\quad$ **if**(root)
16. $\quad\quad$ **if**$(cost\_i < best\_cost)$
17. $\quad\quad\quad best\_cost \leftarrow cost\_i$
18. $\quad\quad\quad best \leftarrow value\_i$
19. $\quad\quad\quad best\_index \leftarrow i$
20. $\quad$ **if** (message from parent includes a new $best\_index\ j$)
21. $\quad\quad best \leftarrow value\_j$
22. $\quad\quad best\_index \leftarrow j$
23. $\quad$ **if** (ReplacementDecision())
24. $\quad\quad$ select and assign the next value
25. $\quad$ delete $value\_(current\_step - (2 * dist))$
26. $\quad$ delete cost of step $(current\_step - height)$
27. $\quad current\_step + +$
28. **for** (1 to $dist + height$)
29. $\quad$ receive message from parent
30. $\quad$ **if** (message from parent includes a new $best\_index\ j$)
31. $\quad\quad best \leftarrow value\_j$
32. $\quad\quad best\_index \leftarrow j$
33. $\quad$ send $best\_index$ to children

Fig. 3. DSA in the ALS_DCOP framework.

step. After every synchronous step the agents can delete the information stored about any of the steps which were not the best and are not of the last $2 * d$ steps. When the algorithm is terminated, the agents must perform another $2h$ steps (again, h is the height of the BFS-tree) in which they do not replace their assignment to make sure that all the agents are aware of the same index of the best step.

The code for DSA in the ALS_DCOP framework is presented in Figure 3 [2]. The structure of the framework is homogeneous for all algorithms with a distributed synchronous local search general structure (such as DSA and DBA). It is interleaved in the algorithm execution as follows:

[2] We assume the existence of a BFS tree when the algorithm begins.

1. In the initialization phase, besides choosing a random value for the variable, agents initialize the parameters which are used by the framework. The root initializes an extra integer variable to hold the cost of the best step (lines 1-7 in Figure 3).
2. In order to get the best out of m steps of the algorithm, $m + h$ steps are performed (notice that for each agent the sum of $height$ and $dist$ is equal to h which is the height of the global BFS-tree). This is required so all the information needed for the root agent to calculate the cost of the m steps will reach it (line 9 in Figure 3).
3. After values are exchanged, each agent calculates the cost of the state according to its height. An agent with height h_i calculates the cost of the state in which its sub-tree was in h_i steps ago. The root agent checks if the cost it calculated is smaller than the best cost found so far and if so saves its information. All other agents check if the best index received from their parent is new. If so they save the information (index and assignment) of the step with the corresponding index (lines 16-22 in Figure 3).
4. Before the step is over, the agent deletes the information that has become redundant. This includes the information on the cost which it passed to its parent on this step and the assignment of the step which its index should have been received on this step in case it was found to be better than previous steps by the root agent (lines 25,26 in Figure 3).
5. On the next step, the value message an agent sends to its parent will include the cost calculation it had performed in this step and the messages to its children will include the index of the best step it knows of.
6. When the termination condition of the algorithm is met, the agents perform additional h steps in which only the best index is propagated down the tree. This way, if the last step cost calculated by the root agent is found to be best, its propagation to all agents is completed. Furthermore, by performing these steps, the possibility that different agents hold best assignments which belong to different steps is prevented (lines 28-33 in Figure 3).

An example of the performance of the ALS_DCOP framework is presented in Figures 4 to 6. To keep the example simple, we only demonstrate the accumulation of the cost of a single step and the propagation of its index once it is found as the best so far. The figures do not show that while the costs of step i are being accumulated, costs and indexes of adjacent steps are also being passed by agents in the BFS-tree.

A $DCOP$ in which the dashed lines connect neighbors in the constraint network and the arrows represent the BFS-tree arcs (each arrow is from parent to child) is presented on the left hand side of Figure 4. The costs in the figure are the private costs calculated for each agent to its state at step i. In the next step, all the leaf agents in the BFS-tree (agents 3, 4 and 5) send their costs to their parents in the tree and the parents add their private costs to the costs they receive from their children. The resulting state is depicted on the right hand side of Figure 4 in which agent 2 added the costs for step i it received from its children agents 4 and 5 to its own cost of step i and got a cost of 8 for step i. Agent 1 received the cost of agent 3 and added it to its own cost but it still did not receive the cost for step i from agent 2. At the next step, agent 1 receives the cost of step i from agent 2 and can calculate the total cost of step i (see the left hand side of Figure 5). Since it is smaller than the best cost achieved so far, agent 1 updates the

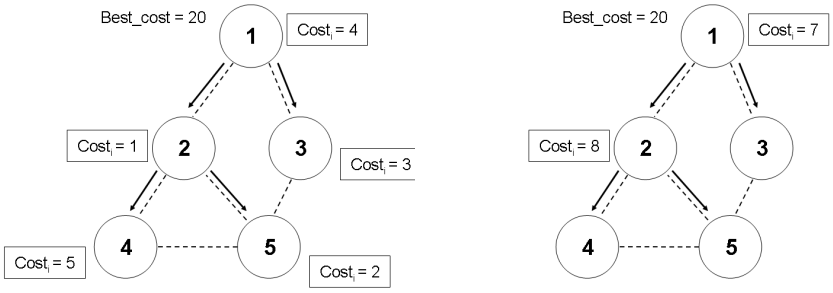

Fig. 4. On the left - Private costs of agents in step i. On the right - Calculations of the cost of step i at step $i + 1$.

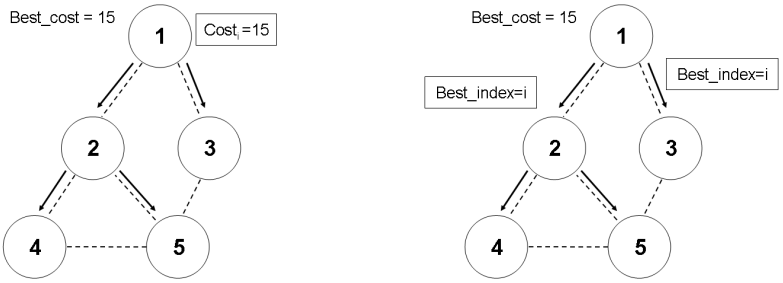

Fig. 5. On the left - Calculations of the cost of step i at step $i + 2$. On the right - Propagation of the new best index, step $i + 3$.

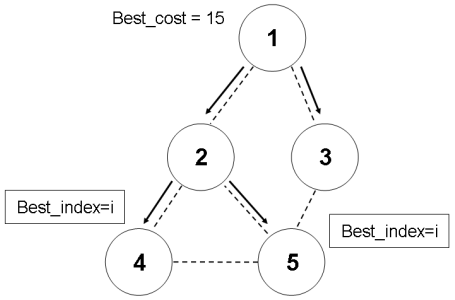

Fig. 6. Propagation of the new best index, step $i + 4$.

new best cost to be 15 and in the next step sends a notification about a new best index in its messages to its children in the BFS-tree (see the right hand side of Figure 5). In the next step (Figure 6), the rest of the agents receive the notification that they should preserve the assignment they held in step i. Since the height of the BFS-tree is 2, the process of accumulating the cost of step i by the root agent and the propagation of the information that it was found to be the best step took 4 steps.

5 Exploration Heuristics

The standard use of local search algorithms for DisCSPs and DCOPs prior to the proposal of the ALS_DCOP framework included running the algorithm for some number of iterations (M) and reporting the complete assignment (solution) held by the agents after the Mth iteration. This use of the algorithm favored exploitive algorithms such as MGM and DSA over explorative algorithms like DBA [22].

The ALS_DCOP framework allows the selection of the best solution traversed by the algorithm and thus can encourage the use of explorative methods. We propose a number of heuristics which implement different approaches towards exploration:

- The first heuristic type we propose combines two exploration strategies which were found successful in previous studies. The first is a periodic increase in the level of exploration for a small number of iterations. This approach was found successful for the DCOP model proposed for mobile sensing agent teams DCOP_MST [25]. The second is periodic restarts which in the case of local search methods result in a selection of a random assignment periodically. The random restart strategy is commonly used in constraint programming methods, e.g., [20]. We used the DSA-C version of DSA as the basic platform on which we incorporated the heuristics which incorporated these two strategies of search. In our basic version of DSA-C an agent replaces its assignment in a 0.4 probability if its best alternative value assignment does not increase the cost of its current assignment. The following two heuristics were combined with DSA-C:
  1. DSA-C-PPIRA1: PPIRA stands for *Periodic Probability Increase and Random Assignments*. Every 15 iterations we increased the probability of replacing an assignment to 0.8 for 5 iterations. Random selections in which each agent selects a random assignment were performed every 35 iterations.
  2. DSA-C-PPIRA2: Every 8 iterations we increased the probability of replacing an assignment to 0.9 for 5 iterations. Random selections in which each agent selects a random assignment were performed every 50 iterations.
- The second exploration approach we implemented formulates a dependency between the probability for replacing an assignment and the potential for improvement that this replacement offers. Such a dependency was suggested for the DSA algorithm in the DSA-B version. I DSA-B agents would not replace assignments if the number of violated constraints was zero. This method is compatible for distributed CSP problems where the aim is to satisfy all constraints. However, it is not applicable for DCOPs for which there is always some endured cost for a pair of assignments of constrained agents. Thus we propose the following heuristic that implements this approach: The heuristic is denoted by DSA-SDP where SDP stands for *Slope Dependent Probability*: If there is an improving alternative the probability for replacing the assignment is calculated as follows:

$$p = 0.65 + min(0.25, \frac{current\_cost - new\_cost}{current\_cost})$$

If the alternative is not improving the current cost the probability is calculated as follows:

$$p = \begin{cases} \frac{current\_cost - new\_cost}{current\_cost} > 1, & 0 \\ \frac{current\_cost - new\_cost}{current\_cost} \leq 1, & max(0.1, 0.4 - \frac{current\_cost - new\_cost}{current\_cost}) \end{cases}$$

In this case (that the best alternative is not improving) we change in probability p only every 40 iterations to allow the algorithm time to converge.

– The last heuristic implements the approach of random restarts in a more monitored way. Here we do not select a complete assignments randomly but rather have single agents select a random assignments when they detect an over exploitive situation. We use the ability of the DBA algorithm to detect *quasi local optima* states and have agents detecting such a situation, break out of them by selecting a random assignment. We call this algorithm DRB which stands for *Distributed Random Breakout*. Like in the DBA algorithm the *quasi local optima* detected by the algorithm are states in which the best alternative of an agent and all its neighbors are not improving the current state. The difference is in the method for escaping this state which is random in DRB in contrast to DBA where the structure of the problem is changed by adding weights to constraints.

6 Experimental Evaluation

In order to emphasize the impact of the ALS_DCOP framework on distributed local search, a set of experiments that demonstrate the effect of the proposed framework when combined with intensive exploration methods is presented.

Three different types of problems were used in the experiments, random DCOPs, graph coloring, and meeting scheduling. These problem were selected to demonstrate the effectiveness of the framework and the proposed heuristics on uniform, structured and realistic problems.

The uniform problems we used in our experiments were minimization random DCOPs in which each agent holds a single variable. Each variable had ten values in its domain. The network of constraints in each of the experiments, was generated randomly by selecting the probability p_1 of a constraint among any pair of agents/variables. The cost of any pair of assignments of values to a constrained pair of variables was selected uniformly between 1 and 10. Such uniform random DCOPs with constraint networks of n variables, k values in each domain, a constraint density of p_1 and a bounded range of costs/utilities are commonly used in experimental evaluations of centralized and distributed algorithms for solving constraint optimization problems [4].

All our experiments on random DCOPs included 120 agents. Each data point represents an average of 50 runs of the algorithm solving different problems.

In our first experiment the probability for a constraint between two agents (density parameter p_1) was set to 0.2. Figure 7 presents the cost of the state in each iteration of the different algorithms. On the left hand side the results presented are for existing local search algorithms for DCOPs. On the right, the results of local search algorithms combined with the exploration methods we propose in this paper are presented. It is quite clear that the trends in these two graphs are very different. On the left hand side we

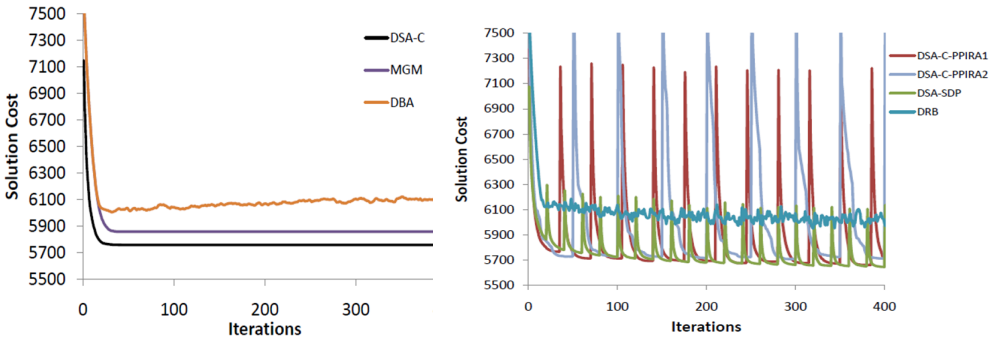

Fig. 7. The cost in each iteration of local search algorithms when solving random DCOPs, $p1 = 0.2$.

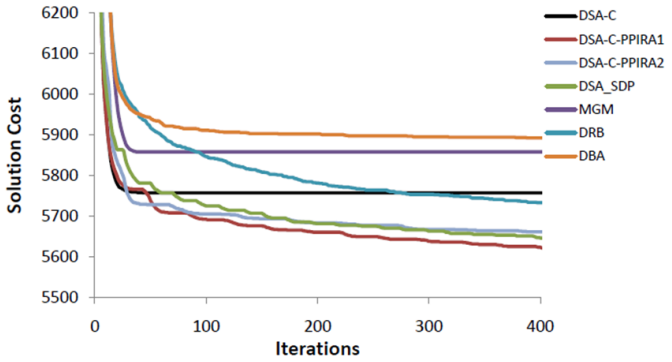

Fig. 8. Anytime cost in each iteration of the local search algorithms when solving random DCOPs, $p1 = 0.2$.

have exploitive algorithms while on the right it is apparent that the algorithm perform intensive exploration.

Figure 8 presents the anytime results for all algorithms on the same random setup. The three exploration heuristics which we combine with DSA outperform the other algorithms. For the DRB algorithm, a larger number of iterations is required (approximately 300) to reduce the cost to a level which is lower than the cost of the solutions found by existing algorithms.

Similar results were obtained for much more dense DCOPs ($p_1 = 0.6$). The results for this setup are presented in Figure 9. While most heuristics perform quite similarly to the more sparse case, the DSA-C-PPIRA2 heuristic has a huge improvement in its anytime performance after 200 iterations. It seems that the extreme explorative selection of parameters allows it to traverse states which no other algorithm explores on dense problems.

The second set of experiments were performed on graph coloring problems. Each graph coloring problem included 120 agents, and as before, each data point represents

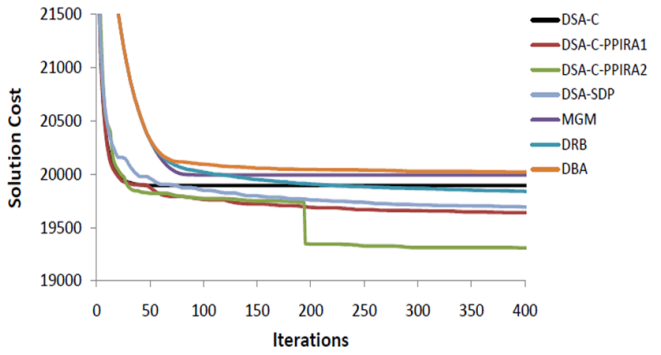

Fig. 9. Anytime cost in each iteration of the local search algorithms when solving random DCOPs, $p1 = 0.6$.

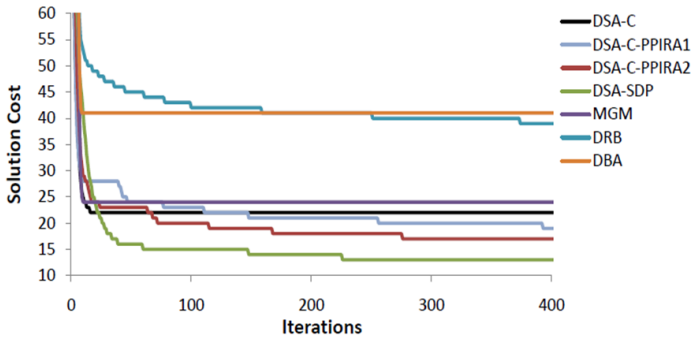

Fig. 10. Anytime cost in each iteration of the local search algorithms when solving graph coloring problems.

an average of 50 runs of the algorithm solving different problems. The number of colors in the problem (domain size) was 3 and the density parameter $p_1 = 0.05$.

Figure 10 presents the results of the algorithms when solving graph coloring problems. Once again the heuristics combined with DSA are most successful. However, the DSA-SDP method outperforms the others. It seems that the structure of the graph coloring problem is exploited best by this method. On the other hand, there is a large gap in between the results obtained by the different versions of DSA and MGM as well to the the results of the two versions of DBA we compared with.

The last set of experiments were performed on realistic *Meeting Scheduling Problems* (MSPs) [3, 9, 10]. The agents' goal in a MSP is to schedule meetings among them. We designed the problem as a minimization problem, thus, agents set their preferences by giving lower costs to meetings which are more important to them. In addition, for every two meetings we selected randomly an *arrival time* that is required to get from

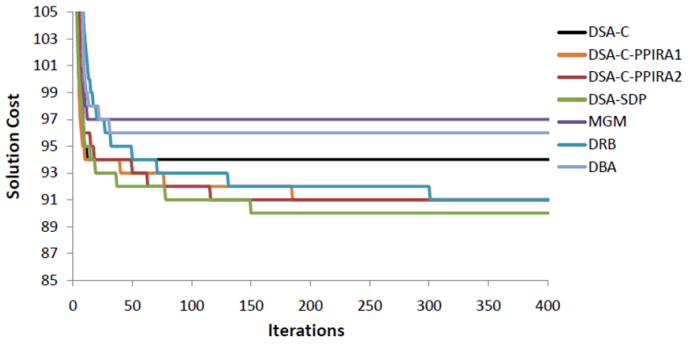

Fig. 11. Anytime cost in each iteration of the local search algorithms when solving m eeting scheduling problems.

the location of one meeting to the other. When the difference between the time-slots of the meetings was less than the arrival time their was a cost endured. The cost was the number of agents participating in these meetings. The setup in this experiment included 90 agents and 20 meetings. There were 20 available time-slots for each meeting. The arrival times between meetings were selected randomly between 6 and 10.

The results in Figure 11 indicate once again that on problems with structure DSA-SDP performs best. On the other hand, in contrast to the graph coloring problems, the DRB algorithm performs well and is competitive with some of the DSA versions.

7 Conclusions

The ALS_DCOP framework enhances synchronous local search algorithms for DCOPs with the *anytime* property. This property enables the use of intensive exploration search methods which were too risky to use before. The negligible cost in time, space and communication load of the framework makes it most attractive. However, existing local search algorithms were designed to converge to a high quality solution at the end of the run and avoided extreme exploration.

In this paper we proposed a set of methods which are substantially more explorative than former local search algorithms. Our experimental evaluation reveals that using these methods within the ALS_DCOP framework improves the quality of the reported solution. Our experimental study included random, structured and realistic problems. It is apparent from our results that explorative heuristics dominate exploitive heuristics on all problem structures. However, for different problem types different exploration heuristics had the advantage.

In future work we intend to investigate the relation between the problem's structure and the heuristic type.

References

1. S. M. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *AAMAS*, pages 1041–1048, 2005.

2. Y. Chong and Y. Hamadi. Distributed log-based reconciliation. In *Proc. ECAI-06*, pages 108–113, August 2006.
3. I.P. Gent and T. Walsh. Csplib: a benchmark library for constraints. Technical report, Technical report APES-09-1999, 1999. Available from http://csplib.cs.strath.ac.uk/. A shorter versionappears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).
4. A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *J. of Artificial Intelligence Research*, 34:25–46, 2009.
5. Fred Glover and Manuel Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.
6. Eliezer Kaplansky and Amnon Meisels. Distributed personnel scheduling - negotiation among scheduling agents. *Annals OR*, 155(1):227–255, 2007.
7. R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *Proc. Parallel and Distributed Computing Systems PDCS)*, pages 432–439, September 2004.
8. Roger Mailer and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. AAMAS-2004*, pages 438–445, July 2004.
9. A. Meisels and O. Lavee. Using additional information in discsp search. In *Proc. 5th workshop on distributed constraints reasoning, DCR-04*, Toronto, 2004.
10. J. Modi and M. Veloso. Multiagent meeting scheduling with rescheduling. In *Proc. of the Fifth Workshop on Distributed Constraint Reasoning (DCR), CP 2004*, Toronto, 2004.
11. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.
12. A. Petcu and B. Faltings. A value ordering heuristic for distributed resource allocation. In *Proc. CSCLP04, Lausanne, Switzerland*, http://liawww.epfl.ch/Publications/Archive/Petcu2004.pdf, 2004.
13. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
14. Andrea Schaerf. Local search techniques for large high-school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics— Part A: Systems and Humans*, 29(4):368–377, 1999.
15. M. C. Silaghi and M. Yokoo. Nogood based asynchronous distributed optimization (adopt ng). In *AAMAS*, pages 1389–1396, 2006.
16. G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (dcsps). In *Constraint Processing-96, (short paper)*, pages 561–2, Cambridge, Massachusetts, USA, October 1996.
17. W. T. L. Teacy, A. Farinelli, N. J. Grabham, P. Padhy, A. Rogers, and N. R. Jennings. Max-sum decentralised coordination for sensor systems. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1697–1698. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
18. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
19. X. Sun W. Yeoh and S. Koenig. Trading off solution quality for faster computation in dcop search algorithms. In *In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 354–360, July 2009.
20. Huayue Wu and Peter van Beek. On universal restart strategies for backtracking search. In *CP*, pages 681–695, 2007.
21. M. Yokoo. Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents & Multi-Agent Sys.*, 3:198–212, 2000.
22. W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.

23. Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.
24. R. Zivan. Anytime local search for distributed constraint optimization. In *AAAI*, pages 393–398, Chicago, IL, USA, 2008.
25. Roie Zivan, Robin Glinton, and Katia Sycara. Distributed constraint optimization for large teams of mobile sensing agents. In *International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 347–354, 2009.