

Evaluation of CBR on Live Networks

Robert N. Lass, Evan A. Sultanik, Pragnesh Jay Modi* and William C. Regli

Drexel University
Department of Computer Science
College of Engineering
3141 Chestnut Street
Philadelphia, PA 19104
`{urlass, eas28, pmodi, regli}@cs.drexel.edu`

Abstract. A large class of problems in multiagent systems can be solved by distributed constraint optimization (DCOP). Several algorithms have been created to solve these problems, however, no extensive evaluation of current DCOP algorithms on live networks exists in the literature. This paper uses DCOPolis—a framework for comparing *and deploying* DCOP software in heterogeneous environments—to contribute an analysis of two state-of-the-art DCOP algorithms run in various network environments solving a number of different problem types. Then, we use this empirical validation to evaluate the use of both cycle-based runtime and concurrent constraint checks.

1 Introduction

A large class of multiagent coordination and distributed resource allocation problems can be modeled as distributed constraint optimization (DCOP) problems. DCOP has generated a lot of interest in the constraint programming community and a number of algorithms have been developed to solve DCOP problems [1–3], however, existing metrics for comparing these algorithms do not adequately capture the many intricacies inherent in solving DCOPs on live networks. This is complicated by the fact that DCOP algorithms are currently implemented in simulation; there is no record in the literature of any significant evaluation of DCOP algorithms on live networks. Furthermore, cycle-based runtime (CBR) metric, for example, has coefficients that are meant to represent network constants, however, no reasonable values for these coefficients are yet known, and the correct values of these coefficients may dictate the ranking of DCOP algorithms. This paper explores when it is useful and when it is not useful to use these metrics.

In the remainder of this paper we first give background on DCOPs and several types of problems we investigate. We then describe a DCOP testbed that we have created to compare algorithms on live networks. We report on a series of experiments run on this testbed using the Adopt [1] and DPOP [3] algorithms. In doing so, we evaluate CBR as a predictor of actual runtime. Finally, we present

* Pragnesh Jay Modi, born 1975, passed away on April 9th, 2007.

an analysis of our results that suggests the coefficients to the CBR equation are actually a function of the algorithm and problem domain, which invalidates CBR (and its special-case *ccc*) as a general metric for comparing DCOP algorithms, but suggest that it may be useful as a metric for predicting asymptotic runtime or even for comparison if the coefficients are known.

1.1 Definitions

Definition 1. A “*DCOP*” is a problem in which a group of agents must distributedly choose values for a set of variables such that the cost of a set of constraints over the variables is either minimized or maximized.

Formally, a DCOP may be represented as a tuple $\langle A, V, \mathcal{D}, f, \alpha, \sigma \rangle$, where:

- A is an ordered set of agents;
- V is an ordered set of variables, $\{v_1, v_2, \dots, v_{|V|}\}$;
- \mathcal{D} is a set of domains, $\{D_1, D_2, \dots, D_{|V|}\}$, where each $D \in \mathcal{D}$ is either a finite or ordered set containing the values to which its associated variable may be assigned;
- f is a function

$$f : \bigcup_{S \in \mathcal{P}(V)} \prod_{v_i \in S} (\{v_i\} \times D_i) \rightarrow \mathbb{N} \cup \{\infty\}$$

(where “ $\mathcal{P}(V)$ ” denotes the power set of V) that maps every possible variable assignment to a cost. This function can also be thought of as defining constraints between variables;

- α is a function $\alpha : V \rightarrow A$ mapping variables to their associated agent. $\alpha(v_i) \mapsto a_j$ implies that it is agent a_j ’s responsibility to assign the value of variable v_i . Note that it is not necessarily true that α is either an injection or surjection; and
- σ is an operator that aggregates all of the individual f costs for all possible variable assignments. This is usually accomplished through summation:

$$\sigma(f) \mapsto \sum_{s \in \bigcup_{S \in \mathcal{P}(V)} \prod_{v_i \in S} (\{v_i\} \times D_i)} f(s).$$

The objective of a DCOP is to have each agent assign values to its associated variables in order to either minimize or maximize $\sigma(f)$.

Definition 2. A “*Context*” is a variable assignment for a DCOP. This can be thought of as a function mapping variables in the DCOP to their current values:

$$t : V \rightarrow (D \in \mathcal{D}) \cup \{\emptyset\}.$$

Note that a context is essentially a partial solution and need not contain values for every variable in the problem; therefore, $t(v_i) \mapsto \emptyset$ implies that the agent $\alpha(v_i)$ has not yet assigned a value to variable v_i . Given this representation, the “domain” (i.e., the set of input values) of the function f can be thought of as

the set of all possible contexts for the DCOP. Therefore, in the remainder of this paper we may use the notion of a context (i.e., the t function) as an input to the f function.

1.2 Examples of DCOP Problems

Graph Coloring Given a graph $G = \langle N, E \rangle$ and a set of colors C , assign each vertex, $n \in N$, a color, $c \in C$, such that the number of adjacent vertices with the same color is minimized. Graph coloring is a commonly-cited problem used for evaluating DCOP algorithms [1, 2].

DCOP Encoding: For each vertex $n_i \in N$, create a variable in the DCOP $v_i \in V$ with domain $D_i = C$. For each pair of adjacent vertices $\langle n_i, n_j \rangle \in E$, create a constraint of cost 1 if both of the associated variables are assigned the same color: $(\forall c \in C : f(\langle v_i, c \rangle, \langle v_j, c \rangle) \mapsto 1)$. A and α cannot be generically defined for graph coloring; they will depend on the application. Most publicly-available benchmark problem sets create one agent per variable [4].

Distributed Multiple Knapsack Problem (DMKP) Given a set of items of varying volume and a set of knapsacks of varying capacity, assign each item to a knapsack such that the amount of overflow is minimized. Let I be the set of items, K be the set of knapsacks, $s : I \rightarrow \mathbb{N}$ be a function mapping items to their volume, and $c : K \rightarrow \mathbb{N}$ be a function mapping knapsacks to their capacities.

DCOP Encoding: for each $i \in I$ create one variable $v_i \in V$ with associated domain $D_i = K$. Then for all possible context t :

$$f(t) \mapsto \sum_{k \in K} \begin{cases} 0 & r(t, k) \leq c(k), \\ r(t, k) - c(k) & \text{otherwise,} \end{cases}$$

where $r(t, k)$ is a function such that

$$r(t, k) = \sum_{v_i \in t^{-1}(k)} s(i).$$

1.3 Evaluation

Cycle-based runtime (CBR) [5], a popular and simple metric used by researchers to evaluate DCOP algorithms, is evaluated in this section. The focus was chosen to be on CBR (over other metrics such as non-concurrent constraint checks [6]) since CBR and its special-case *ccc* are the metrics most often employed in evaluating DCOP algorithms in the literature [1, 3, 2].

2 Experimental Setup

2.1 Software

The reference implementations for the Adopt and DPOP algorithms (coded by their respective authors) were designed to be run in simulation; although extending the code to be run on a live network was not hard, configuring it for

automated batch processing of experiments in such a setting was non-trivial. Therefore, the implementations of these algorithms as provided in the DCOPolis¹ package were used.

DCOPolis was chosen as the testbed for our experiments because it was originally designed as framework for comparing and deploying distributed decision processes in heterogeneous environments. At the time the experiments were performed, DCOPolis had three DCOP algorithms implemented: Adopt, DPOP and a naïve algorithm called Distributed Hill Climbing. Only Adopt and DPOP were used for our experiments.

DCOPolis differentiates itself from existing frameworks and simulators (like FRODO [7] and those used in testing Adopt and OptAPO) in two fundamental ways:

1. DCOPolis was designed to allow for both simulation of DCOPs on a single computer and full deployment of DCOP solvers on many types of live networks, including traditional wired networks and ad-hoc wireless networks; and
2. DCOPolis is able to instantiate a DCOPs and start the solution process completely distributedly. This means that there is no need for configuration files, nor is there any need for a central agent/server that initializes/instantiates the rest of the group.

All of the code is freely available under the GNU public license.

2.2 Pseudotree Generation

A similarity between Adopt and DPOP is that they both assume the existence of a tree ordering over all of the variables in the problem. The pseudotree has an invariant that for each pair of variables $\langle v_i, v_j \rangle$ that are neighboring in the constraint graph it must be the case that v_i is either an ancestor or descendent of v_j in the pseudotree. The pseudotree also contains a backedge between all pairs of neighbors in the constraint graph that do not have a parent/child relationship in the pseudotree. For each $v \in V$, $\alpha(v)$ must know the relative tree position (*i.e.*, ancestor, descendent, or parent) of each constraint graph neighbor of v . The authors of both Adopt and DPOP assume that the agents would simply elect one agent to create this ordering which is then broadcast to the rest of the group. Since the runtime of both algorithms is highly dependent on the structure of the pseudotree, we ensured that for each problem instance in our experiments the algorithms were given identical pseudotrees.

2.3 Computing Devices

Five HP-TC4200 tablet PCs with 1.73Ghz Intel Pentium M processors and 512M of RAM were connected via Ethernet to a Netgear FS108 switch. No machines were connected to the switch other than the ones taking part in the experiment

¹ <http://dcopolis.sourceforge.net/>

and the switch was not connected to the Internet or any other network. All the machines were running Ubuntu 7.04 Linux with a 2.6.22 kernel. For the wireless experiments, the computers communicated over a private ad-hoc 802.11g network.

2.4 Problem Datasets

Multiagent Task Scheduling Experiments on the data multiagent task scheduling C_TÆMS dataset referenced in [8] was attempted, however, DPOP was unable to solve any of these problems. This was likely due to the problems' large number of variables and domain sizes. This is analyzed in §4.1.

Graph Coloring The USC Teamcore project has a variety of sample problem data files in their DCOP repository [4] which were used in our analysis. The graph coloring problems were from the “Graph coloring dataset” and range from 8 to 30 variables. The dataset is composed of 3-coloring problems (*i.e.*, problems in which one of three colors must be assigned to each vertex). The problems are encoded as above (*i.e.*, there is one variable for each vertex and the variables' domains are the set of possible colors). In the interest of experimentally analyzing the effect of domain size, we augmented the dataset by also solving 4- and 5-coloring problems.

Distributed Multiple Knapsack Problem DCOPolis has a utility for creating random DMKP data files. Twenty-five problem sets were created, consisting of five of each of the following: many small bins (ten), many small objects (twelve); few small bins (three), many large objects; few large bins, many small objects; few small bins, wide variety (high standard deviation) of objects and a wide variety of bins, many small objects. These data files are available from the authors' website.

2.5 Cycle-based Metrics

In the first publication introducing Distributed Constraint Satisfaction, [9], Yokoo, *et al.* evaluate algorithms by counting the number of cycles needed to determine a solution. A cycle is a block of computation time between sending messages. The cost of communications is not taken into account, which the authors note and explain by stating that they do not have a standard way to compare communication costs and computational costs.

Cycle-based runtime (CBR) was introduced in [5] as a metric that takes into account the number of constraint checks performed in each cycle as well as the communications latency between cycles. CBR is computed as

$$CBR(m) = L \times m + ccc(m) \times t,$$

where t and L are constants respectively relating to overhead in computation and communication, m is the number of cycles, and

$$ccc(m) = \sum_{k=0}^m \max_{a \in A} cc(a, k),$$

where $cc(a, k)$ is the number of constraint checks performed by agent a in cycle k .

Given the fact that a single host on the network can support multiple agents (and assuming that each host has a single processor), CBR must take into account the number of machines used in the solution of the DCOP. Therefore, we propose a slight modification to CBR that accounts for the distribution of agents on the hosts:

$$ccc(m) = \sum_{k=0}^m \max_{h \in H} \sum_{a \in A_h} cc(a, k),$$

where H is the set of all hosts and A_h is the set of agents on host h . In other words, all agents that are running on the same host must compete for time from the single CPU, so these agents are in effect running synchronously during each cycle. Therefore, for all agents that are sharing a host we need to sum over the number of constraint checks during each cycle instead of taking the maximum. Given an experiment where $\max_{h \in H} |A_h| = 1$ (which implies $|H| \geq |A|$), the two equations are equivalent. For the remainder of the paper we shall use this augmented definition of CBR.

When calculating CBR outside of simulation, one has to take into account that agents running on different physical computers will have no way of synchronizing their cycle counters. For example, consider a network of n computers with one agent running on each computer. Agent a_1 sends message m_1 to agent a_2 , who, in response, sends message m_2 to a_3 . This trend continues through all of the agents: agent a_i sends message m_i to agent a_{i+1} in response to receiving message m_{i-1} from agent a_{i-1} . Finally, when agent a_n receives message m_{n-1} from agent a_{n-1} , it sends message m_n back to the first agent: a_1 . During this propagation of the message through the network, agent a_1 sent another message m' to a_2 . Since a_n has only sent one message (namely m_n), a_n will believe that it sent m_n during cycle 2, however, since a_1 has sent two messages it will believe that it received m_n during cycle 3. This is a contradiction, and necessitates distributed synchronization of agents' cycles.

DCOPolis implements distributed synchronization of cycles by appending to each inter-agent message the cycle during which it was sent. When an agent receives a message, the agent increments its cycle counter until its cycle is greater than the cycle during which the message was sent. The overhead from adding the cycle during which each message was sent is marginal. This method is somewhat similar to that proposed in [6].

3 Results and Analysis

The results of the graph coloring experiments on both a wired and wireless network can be seen in Figures 1 and 2 respectively. The DMKP experiments

can be seen in Figure 3. In all three graphs, Adopt and DPOP both show a linear correlation between runtime and CBR. In Figure 4, the results of running graph coloring problems with large domains and fifty sparsely connected vertices is shown. DPOP was unable to solve many of these problems due to the algorithm running out of memory while trying to construct the massive hypercubes for this problem domain. There are DPOP variants[10,11] that may scale better, but they are not yet implemented in DCOPolis.

Pearson's linear correlation coefficient was calculated for the runtime and the CBR metric. For each of the datasets except two we were able with 99% certainty to reject the null hypothesis that the distributions were not linearly correlated in favor of the alternate hypothesis that CBR and actual runtime are linearly correlated. Pearson's coefficient has a student's t distribution, which is what we used to test these hypotheses. Among the data for which we could reject the null hypothesis with 99% certainty, our smallest test statistic value was 3.05. We were only able to reject the null hypothesis with 78% certainty for the DPOP data in Figure 2 (test statistic: 1.26). The one test for which we failed to reject the null hypothesis with reasonable certainty was for the DPOP data in Figure 3. It is clear from looking at the graph, however, that the large cluster of DPOP data supports the claim that CBR is a valid metric for predicting actual runtime.

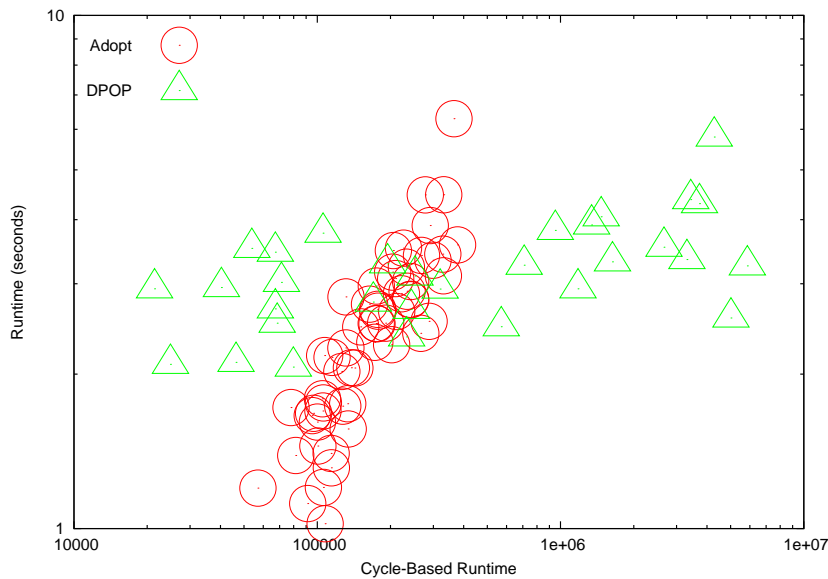


Fig. 1. Actual runtime versus cycle-based runtime the graph coloring problem set running on a wired 100BaseT network. Both Adopt and DPOP exhibit a linear correlation. Both axes are scaled logarithmically in order to reduce clustering around the origin.

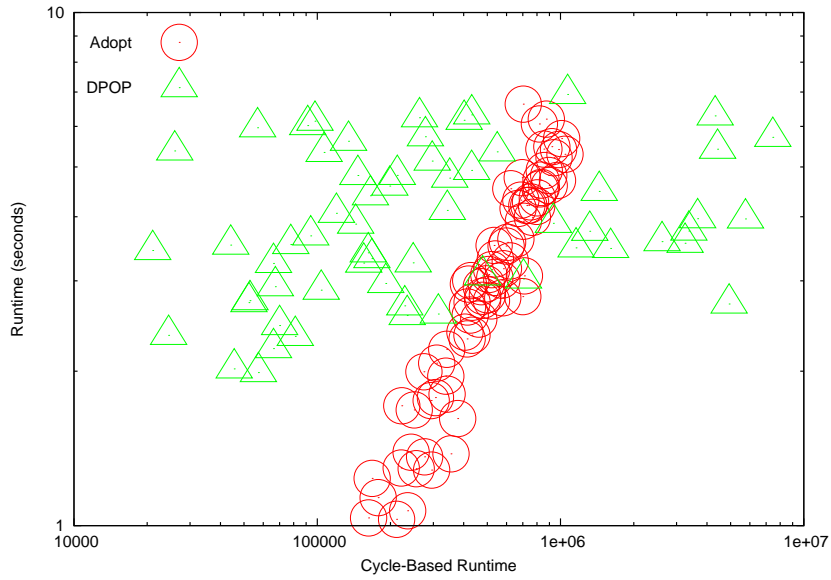


Fig. 2. Actual runtime versus cycle-based runtime the graph coloring problem set running on a wireless ad-hoc network. Both Adopt and DPOP exhibit a linear correlation. Both axes are scaled logarithmically in order to reduce clustering around the origin.

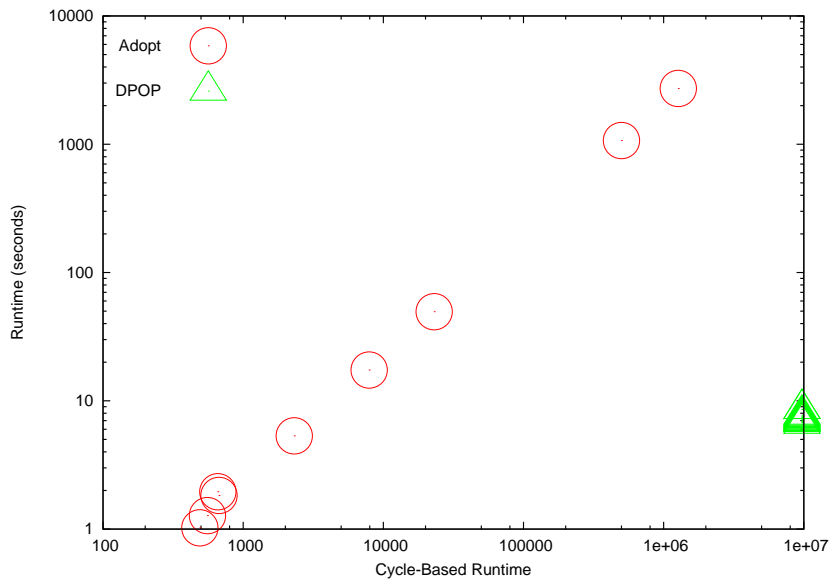


Fig. 3. Actual runtime versus cycle-based runtime for a randomly-generated set of DMK problems. Both the number of knapsacks and number of items were varied. Both Adopt and DPOP exhibit a linear correlation. Both axes are scaled logarithmically in order to reduce clustering around the origin.

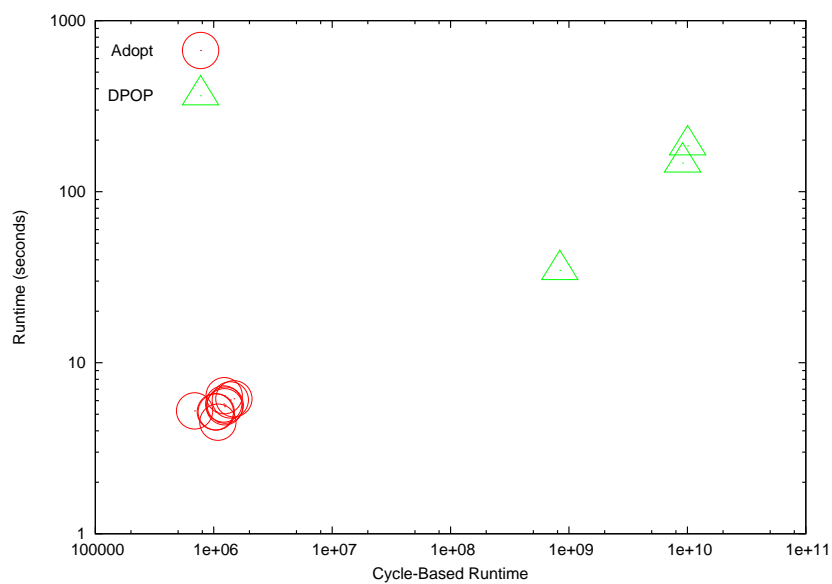


Fig. 4. Actual runtime versus cycle-based runtime for a randomly-generated set of eight-color graph coloring problems with fifty vertices. There are only three DPOP data points; the other seven failed due to a lack of memory. Both Adopt and DPOP exhibit a linear correlation. Both axes are scaled logarithmically in order to reduce clustering around the origin.

L and t were calculated empirically for each of the domains and algorithms. The average time spent sending and receiving data during each cycle was calculated and used as L . The average runtime per cycle—not counting time required for communication—was used as t . As shown in Table 1, these coefficients were quite different between algorithms and problem domains. The coefficients calculated in the graph coloring domain had reasonably similar t values, but reasonably different L values. This was expected, given that the computational power available was the same, but the communications environment was less efficient.

PROBLEM DOMAIN	NETWORK	ALGORITHM	L	t
Graph Coloring	Wired	Adopt	1572.97	28.07
		DPOP	1.27	166.02
	Wireless	Adopt	4988.3	25.55
		DPOP	2.12	163.08
DMKP	Wired	Adopt	54.01	68.62
		DPOP	215.78	4299.18

Table 1. Empirically-determined values for the CBR coefficients as a function of the problem domain, network setting, and DCOP algorithm.

4 Conclusions and Future Work

We have shown that CBR is an excellent predictor of asymptotic runtime. We have also shown that the L and t coefficients in the CBR metric are not in fact constant, even when the network environment is constant. These coefficients are best represented as a function of the algorithm and the problem domain, and it is currently unclear how these can be predicted through traditional simulation. CBR therefore falls short as a metric for comparing algorithms, unless the coefficients for each algorithm are known *a priori*. We have provided a list of these coefficients for a number of different problems. In the future we hope to expand this list and also investigate new metrics such as non-concurrent constraint checks [6].

The runtime of these algorithms is highly dependent on the variable ordering given by the pseudotree. Our next experiments will be to measuring the impact of alternate techniques for generating these trees, such as [12].

DCOPolis supports the use of the Sefirs² simulation kernel and MATES network simulator [13], which essentially creates a virtual machine that runs in simulated time. We hope to use our live network data to calibrate these simulations to allow for a comparison of DCOP algorithms empirically in *simulation*, without the need for theoretical comparison metrics like CBR or access to a cluster of computers or testbed like the one created for this paper.

² <http://sefirs.sourceforge.net/>

4.1 A note on comparisons

It is not the authors' intent to directly compare the algorithmic performance of Adopt and DPOP in this paper. The reference implementations for these algorithms (coded by their respective authors) were designed to be run in simulation; although extending the code to be run on a live network was not hard, configuring it for automated batch processing of experiments in such a setting was non-trivial. Therefore, the implementations of these algorithms as provided in the DCOPolis package were used. These implementations were created by authors other than the original algorithm designers, based solely upon the algorithms described in the respective papers. Furthermore, there are other techniques and variations of both Adopt [14–16] and DPOP [17, 11, 18–23] that may have performed differently given our experimental datasets.

Although the data in this paper seem to suggest DCOPolis' implementation of DPOP outperforms ADOPT in terms of runtime, they are insufficient to objectively declare DPOP a better algorithm. The favorable runtimes of DPOP may be due to our selection of small problems; larger problems (*e.g.*, coloring problems with large domains and C.TEMS problems) cannot be run with DCOPolis' implementation of DPOP because the hypercubes DPOP generates require far too much memory. DPOP's worst-case memory usage scales exponentially with respect to the average domain size [3], while Adopt scales polynomially [1]. For example, Figure 4 shows a graph of experiments that used randomly generated graph coloring problems of fifty sparsely connected vertices using eight colors. Of the ten experiments, only three completed for the DPOP algorithm; the other eight failed due to the inability to allocate enough memory. All of the Adopt problems finished.

References

1. 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, New York, NY, USA, ACM Press (2003) 161–168
2. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 438–445
3. Petcu, A., Faltings, B.: A distributed, complete method for multi-agent constraint optimization. In: CP 2004 - Fifth International Workshop on Distributed Constraint Reasoning (DCR2004), Toronto, Canada (September 2004)
4. Pearce, J.P.: University of southern california DCOP repository (2007) <http://teamcore.usc.edu/dcop/>.
5. Davin, J., Modi, P.J.: Impact of problem centralization in distributed constraint optimization algorithms. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2005) 1057–1063
6. Meisels, A., Kaplansky, E., Razgon, I., Zivan, R.: Comparing performance of distributed constraints processing algorithms (2002)

7. Petcu, A.: Frodo: A framework for open/distributed constraint optimization. Technical Report No. 2006/001 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland) (2006) <http://liawwww.epfl.ch/frodo/>.
8. Sultanik, E., Modi, P.J., Regli, W.C.: On modeling multiagent task scheduling as a distributed constraint optimization problem. In: IJCAI. (2007) 1531–1536
9. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering* **10**(5) (1998) 673–685
10. Petcu, A., Faltings, B.: Mb-dpop: A new memory-bounded algorithm for distributed optimization. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07, Hyderabad, India (Jan 2007) 1452–1457
11. Petcu, A., Faltings, B., Parkes, D.: M-DPOP: Faithful distributed implementation of efficient social choice problems. submitted to the *Journal of Artificial Intelligence Research (JAIR)* (2007) submitted.
12. Chechetka, A., Sycara, K.: A decentralized variable ordering method for distributed constraint optimization. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2005) 1307–1308
13. Sultanik, E.A., Peysakhov, M.D., Regli, W.C.: Agent transport simulation for dynamic peer-to-peer networks. Technical Report DU-CS-04-02, Drexel University (2004)
14. Davin, J.P.: Algorithmic and domain centralization in distributed constraint optimization problems Master's Thesis, CMU-CS-05-154, CMU Tech Report, 2005.
15. Silaghi, M.C., Yokoo, M.: Nogood based asynchronous distributed optimization (adopt ng). In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 1389–1396
16. Ali, S., Koenig, S., Tambe, M.: Preprocessing techniques for accelerating the dpop algorithm adopt. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2005) 1041–1048
17. Petcu, A., Faltings, B., Parkes, D., Xue, W.: BB-M-DPOP: Structural techniques for budget-balance in distributed implementations of efficient social choice. Technical report id: Lia-report-2007-002, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland) (April 2007)
18. Kumar, A., Petcu, A., Faltings, B.: H-DPOP: Using hard constraints to prune the search space. In: IJCAI'07 - Distributed Constraint Reasoning workshop, DCR'07. (Jan 2007)
19. Petcu, A., Faltings, B., Mailler, R.: Pc-dpop: A new partial centralization algorithm for distributed optimization. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07, Hyderabad, India (Jan 2007) 167–172
20. Petcu, A., Faltings, B.: S-dpop: Superstabilizing, fault-containing multiagent combinatorial optimization. In: Proceedings of the National Conference on Artificial Intelligence, AAAI-05, Pittsburgh, Pennsylvania, USA, AAAI (July 2005) 449–454
21. Petcu, A., Faltings, B.: R-dpop: Optimal solution stability in continuous-time optimization. In: IJCAI 2005 - DCR Workshop (Distributed Constraint Reasoning), Edinburgh, Scotland (Aug 2005)
22. Petcu, A., Faltings, B.: Ls-dpop: A propagation/local search hybrid for distributed optimization. In: CP 2005- LSCS'05: Second International Workshop on Local Search Techniques in Constraint Satisfaction, Sitges, Spain (October 2005)

23. Petcu, A., Faltings, B.: O-dpop: An algorithm for open/distributed constraint optimization. In: Proceedings of the National Conference on Artificial Intelligence, AAAI-06, Boston, USA (July 2006) 703–708