Trading Off Solution Cost for Smaller Runtime in DCOP Search Algorithms (Extended Version)

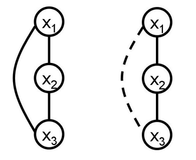
William Yeoh   Sven Koenig   Xiaoxun Sun

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA
{wyeoh,skoenig,xiaoxuns}@usc.edu

Abstract. Distributed Constraint Optimization (DCOP) is a key technique for solving multiagent coordination problems. Unfortunately, finding minimal-cost DCOP solutions is NP-hard. We therefore propose two mechanisms that trade off the solution costs of two DCOP search algorithms (ADOPT and BnB-ADOPT) for smaller runtimes, namely the Inadmissible Heuristics Mechanism and the Relative Error Mechanism. The solution costs that result from these mechanisms are bounded by a more meaningful quantity than the solution costs that result from the existing Absolute Error Mechanism since they both result in solution costs that are larger than minimal by at most a user-specified percentage. Furthermore, the Inadmissible Heuristics Mechanism experimentally dominates both the Absolute Error Mechanism and the Relative Error Mechanism for BnB-ADOPT and is generally no worse than them for ADOPT.

1 Introduction

Distributed Constraint Optimization (DCOP) is a key technique for solving multiagent coordination problems. Researchers have therefore developed a variety of DCOP search algorithms [6, 9, 2, 3], including ADOPT [8] and BnB-ADOPT [11]. ADOPT is one of the pioneering DCOP search algorithms that uses a best-first search strategy. BnB-ADOPT is a variant of ADOPT that uses a depth-first branch-and-bound search strategy and has been shown to be faster than ADOPT on several DCOP problems. Finding minimal-cost DCOP solutions is NP-hard [8], which makes it advantageous to allow users to trade off between the runtimes of DCOP algorithms and the resulting solution costs. ADOPT is, to the best of our knowledge, the only DCOP algorithm with this property. Its Absolute Error Mechanism allows its users to specify the absolute errors on the solution costs, for example, that the solution costs should be at most 10 larger than minimal. However, it is often much more meaningful for users to specify the relative errors on the solution costs, for example, that the solution costs should be at most 10 percent larger than minimal, which cannot be done with the Absolute Error Mechanism without knowing the minimal solution costs a priori. In this



x_1	x_2	Cost
0	0	3
0	1	8
1	0	20
1	1	3

x_1	x_3	Cost
0	0	5
0	1	4
1	0	3
1	1	3

x_2	x_3	Cost
0	0	3
0	1	8
1	0	10
1	1	3

(c)

Fig. 1. Example DCOP Problem

paper, we propose two mechanisms with this property, namely the Relative Error Mechanism and the Inadmissible Heuristics Mechanism, and show that the Inadmissible Heuristics Mechanism experimentally dominates both the Absolute Error Mechanism and the Relative Error Mechanism for BnB-ADOPT and is generally no worse than them for ADOPT.

2 DCOP Problems

A DCOP problem is given by a finite number of variables with their finite domains and a finite set of constraints. Each constraint involves two variables and specifies a non-negative constraint cost as a function of the values of these two variables. A solution assigns each variable a value from its domains, while a partial solution might not assign values to all variables. The cost of a solution is the sum of the constraint costs of all constraints.

DCOP problems can be represented with constraint graphs, whose vertices are the variables and whose edges are the constraints. ADOPT and BnB-ADOPT transform constraint graphs in a preprocessing step into constraint trees. Constraint trees are spanning trees of constraint graphs with the property that edges of the constraint graphs can connect vertices only with their ancestors or descendants in the constraint trees. Figure 1(a) shows the constraint graph of an example DCOP problem with three variables that can each be assigned either zero or one, and Figure 1(c) shows the constraint costs. Figure 1(b) shows one possible constraint tree. The dotted line is part of the constraint graph but not the constraint tree.

3 Search Trees and Heuristics

The operation of ADOPT and BnB-ADOPT can be visualized with AND/OR search trees [7]. We use regular search trees and terminology from A* [4] for our example DCOP problem since its constraint tree is a chain. We refer to its nodes with the identifiers shown in Figure 2(a). Its layers correspond to the variables. A left branch that enters a layer means that the corresponding variable is set to zero, and a right branch means that the corresponding variable is set to one. For our example DCOP problem, the partial solution of node e is $(x_1 = 0, x_2 = 1)$. The f^*-value of a node is the minimal cost of any solution that completes the

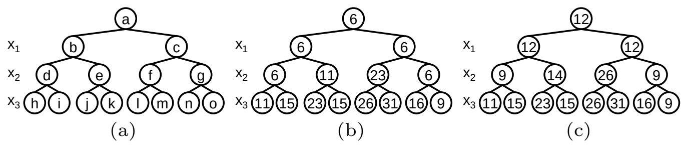


Fig. 2. Search Trees for the Example

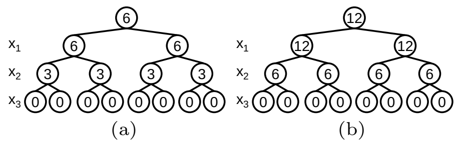


Fig. 3. h-Values for the Example

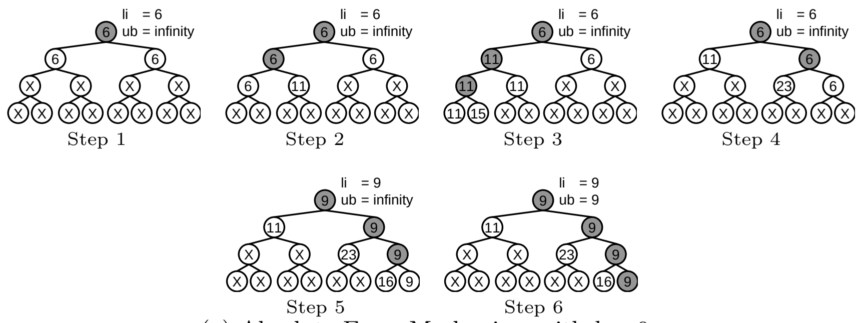
partial solution of the node. For our example DCOP problem, the f^*-value of node e is the minimum of the cost of solution $(x_1 = 0, x_2 = 1, x_3 = 0)$ [=23] and the cost of solution $(x_1 = 0, x_2 = 1, x_3 = 1)$ [=15]. Thus, the f^*-value of node e is 15. The f^*-value of the root node is the minimal solution cost.

ADOPT and BnB-ADOPT use estimated f^*-values, called f-values, during their searches since the f^*-values are not known a priori. They calculate the f-values from user-specified h-values (heuristics), as follows: ADOPT and BnB-ADOPT calculate the f-value of a node by summing the costs of all constraints that involve two variables with known values and adding an h-value that estimates the sum of the unknown costs of the remaining constraints, similarly to how A* calculates the f-values of its nodes. For our example DCOP problem, if the h-value of node e is 3, then its f-value is 11, namely the sum of the known cost of the constraint between variables x_1 and x_2 [=8], and its h-value. The ideal h-values result in f-values that are equal to the f^*-values. For our example DCOP problem, the ideal h-value of node e is $15 - 8 = 7$. Admissible h-values do not overestimate the ideal h-values.

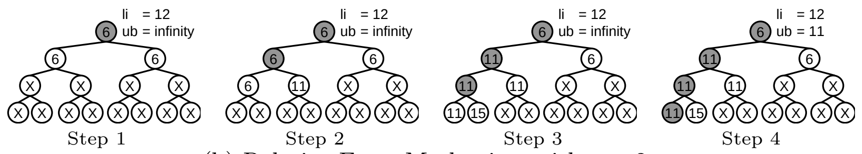
4 ADOPT and BnB-ADOPT

We now give an *extremely simplistic* description of the operation of ADOPT and BnB-ADOPT to explain the search principles behind them. For example, we assume that information propagation is instantaneous. Complete descriptions of ADOPT and BnB-ADOPT can be found in [8, 11].

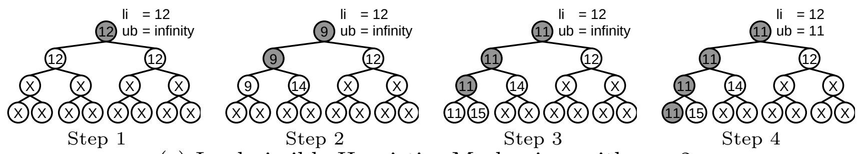
ADOPT originally used zero h-values but was later extended to use admissible h-values. BnB-ADOPT was directly designed to use admissible h-values. We thus assume for now that the h-values are admissible (to be precise: consistent). For our example DCOP problem, we use the h-values from Figure 3(a), which result in the f-values from Figure 2(b).



(a) Absolute Error Mechanism with $b = 0$



(b) Relative Error Mechanism with $p = 2$



(c) Inadmissible Heuristics Mechanism with $c = 2$

Fig. 4. Execution Traces of ADOPT

We visualize the operation of ADOPT and BnB-ADOPT with search trees, as shown in Figures 4 and 5. The nodes that are being expanded and their ancestors are shaded grey. ADOPT and BnB-ADOPT maintain lower bounds for all grey nodes and their children, shown as the numbers in the nodes. ADOPT and BnB-ADOPT initialize the lower bounds with the f-values and then always set them to the minimum of the lower bounds of the children of the nodes. Memory limitations prevent them from maintaining the lower bounds of the other nodes, shown with crosses in the nodes. ADOPT and BnB-ADOPT also maintain upper bounds, shown as ub. They always set them to the smallest costs of any solutions found so far. Finally, ADOPT maintains limits, shown as li.[1] It always set them to b plus the maximum of the lower bounds $lb(r)$ and the f-values $f(r)$ of the root nodes $[li = b + \max(lb(r), f(r))]$, where $b \geq 0$ is a user-specified absolute error bound. To maintain consistency, we extend BnB-ADOPT to maintain limits with the exact same definition as above.

ADOPT expands nodes in a depth-first search order (it always expands the child of the current node with the smallest lower bound) and backtracks when

[1] li is similar to the threshold of the root node in ADOPT.

the lower bounds of all unexpanded children of the current node are larger than the limits. This search order is identical to a best-first search order, where one ignores the nodes with crosses and always expands the fringe nodes with the smallest lower bounds next. BnB-ADOPT also expands nodes in a depth-first search order (it always expands the child of the current node with the smallest lower bound) but backtracks when the lower bounds of all unexpanded children of the current node are no smaller than the upper bounds.

ADOPT and BnB-ADOPT terminate once the limits (that are equal to b plus the largest known lower bounds on the minimal solution costs) are no smaller than the upper bounds (that are equal to the smallest known upper bounds on the minimal solution costs) [$li \geq ub$].[2] Thus, ADOPT and BnB-ADOPT should terminate with solution costs that are at most b larger than minimal (a fact currently proved only for ADOPT), which is why we refer to this mechanism as the Absolute Error Mechanism.

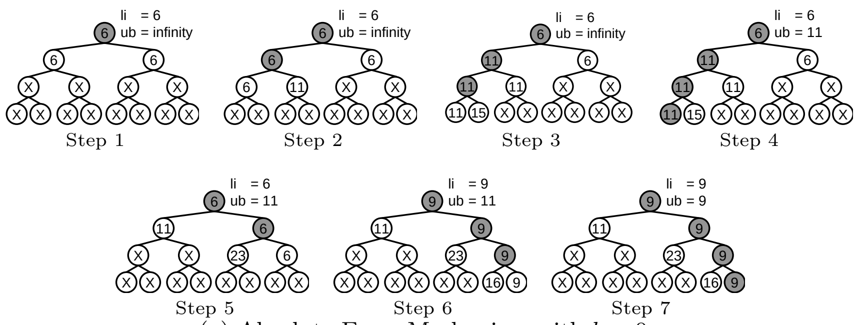
Figures 4(a) and 5(a) show execution traces of ADOPT and BnB-ADOPT, respectively, with the Absolute Error Mechanism with $b = 0$ for our example DCOP problem, that is, finding the minimal solution costs.

5 Relative Error Mechanism

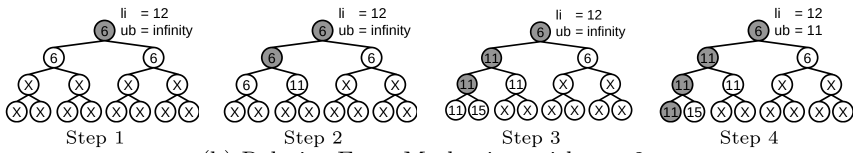
We argued that it is often much more meaningful for users to specify the relative error on the solution costs than the absolute errors, which cannot be done with the Absolute Error Mechanism without knowing the minimal solution costs a priori. However, we can easily change the Absolute Error Mechanism of ADOPT and BnB-ADOPT to a Relative Error Mechanism. ADOPT and BnB-ADOPT now set the limits to p times the maximum of the lower bounds $lb(r)$ and the f-values $f(r)$ of the root nodes [$li = p \times \max(lb(r), f(r))$], where $p \geq 1$ is a user-specified relative error bound. ADOPT and BnB-ADOPT still terminate once the limits (that are now equal to p times the largest known lower bounds on the minimal solution costs) are no smaller than the upper bounds (that are equal to the smallest known upper bounds on the minimal solution costs). Thus, they should terminate with solution costs that are at most p times larger than minimal (a fact currently unproved) or, equivalently, at most $(p-1) \times 100$ percent larger than minimal, which is why we refer to this mechanism as the Relative Error Mechanism. The guarantee of the Relative Error Mechanism with relative error bound p is thus similar to the guarantee of the Absolute Error Mechanisms with b equal to $p - 1$ times the minimal solution cost, except that the user does not need to know the minimal solution costs a priori.

Figures 4(b) and 5(b) show execution traces of ADOPT and BnB-ADOPT, respectively, with the Relative Error Mechanism with $p = 2$ for our example DCOP problem. For example, after ADOPT expands node d in Step 3, the lower bound [=11] of unexpanded child h of node e is no larger than the limit [=12]. ADOPT thus expands the child [=h] with the smallest lower bound in Step 4.

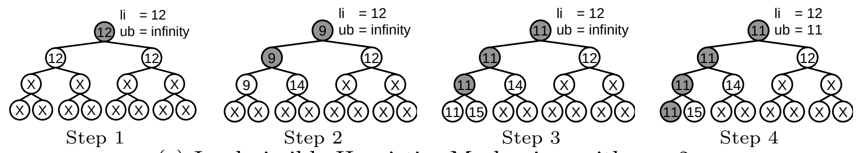
[2] The unextended BnB-ADOPT terminates when $lb(r) = ub$.



(a) Absolute Error Mechanism with $b = 0$



(b) Relative Error Mechanism with $p = 2$



(c) Inadmissible Heuristics Mechanism with $c = 2$

Fig. 5. Execution Traces of BnB-ADOPT

The limit is now no smaller than the upper bound and ADOPT terminates. However, after ADOPT in Figure 4(a) expands node d in Step 3, the lower bounds of all unexpanded children of node d are larger than the limit. ADOPT backtracks repeatedly and expands node c next and terminates eventually in Step 6. Thus, ADOPT with the Relative Error Mechanism with $p = 2$ terminates two steps more quickly than in Figure 4(a) since it switches contexts less often, but with a solution cost that is 2 larger.

6 Inadmissible Heuristics Mechanism

The h-values should be close to the ideal h-values since runtimes of search methods are then small. One therefore often multiplies admissible h-values of A* with a constant $c \geq 1$, which trades off between the runtime of A* and the resulting path length. The resulting paths are at most c times larger than minimal [10]. We let ADOPT and BnB-ADOPT use the exact same mechanism. We therefore assume that the h-values are derived from admissible (to be precise: consistent) h-values by multiplying them with a constant $c \geq 1$, which can result in inadmissible h-values. For our example DCOP problem, we use the h-values from Figure

3(b), which are derived from the admissible h-values by multiplying them with 2 and result in the f-values from Figure 2(c). We also assume that ADOPT and BnB-ADOPT use no error bounds, that is, either the Absolute Error Mechanism with $b = 0$ or the Relative Error Mechanism with $p = 1$. Then, ADOPT and BnB-ADOPT terminate once the lower bounds of the root nodes (that can now be at most c times larger than the minimal solution costs and thus are no longer lower bounds on the minimal solution costs, despite their name) are no smaller than the upper bounds (that are equal to the smallest known upper bounds on the minimal solution costs). Thus, ADOPT and BnB-ADOPT should terminate with solution costs that are at most c times larger than minimal (a fact currently unproved). Therefore, the Inadmissible Heuristics Mechanism shares the advantages of the Relative Error Mechanism.

Figures 4(c) and 5(c) show execution traces of ADOPT and BnB-ADOPT, respectively, with the Inadmissible Heuristics Mechanism with $c = 2$ for our example DCOP problem. ADOPT terminates two steps more quickly than in Figure 4(a) since it switches contexts less often, but with a solution cost that is 2 larger.

7 Experimental Results

We now compare ADOPT and BnB-ADOPT with the Absolute Error Mechanism, the Relative Error Mechanism and the Inadmissible Heuristics Mechanism in three domains, using DP2 as admissible h-values [1]:

- For **graph coloring**, the variables are the vertices, their domains are the colors, and the constraints are between adjacent vertices [8]. We fix the number of vertices to 10 and the density, defined as the ratio of the number of constraints and the number of vertices, to 2. Each variable always has three possible values. All costs are randomly generated from 0 to 10,000. We average the experimental results over 50 DCOP instances with randomly generated constraints.
- For **sensor network scheduling**, the variables are the targets that need to be tracked, their domains are the time slots during which they could be tracked, and the constraints are between adjacent targets [5]. We fix the number of targets to 9. Each target always has eight possible time slots. The cost of assigning a time slot to a target that is also assigned to an adjacent target is infinity (to be precise: 1,000,000) since any one sensor cannot track two targets during the same time slot. The cost of targets that are not tracked during any time slot is 100. All other costs are randomly generated from 0 to 100. We average the experimental results over 50 DCOP instances.
- For **meeting scheduling**, the variables are the meetings that need to be scheduled, their domains are the time slots in which they could be scheduled, and the constraints are between meetings that share participants [5]. We fix the number of meetings to 5. Each meeting always has eight possible time slots. The cost of assigning a time slot to a meeting that has at least one

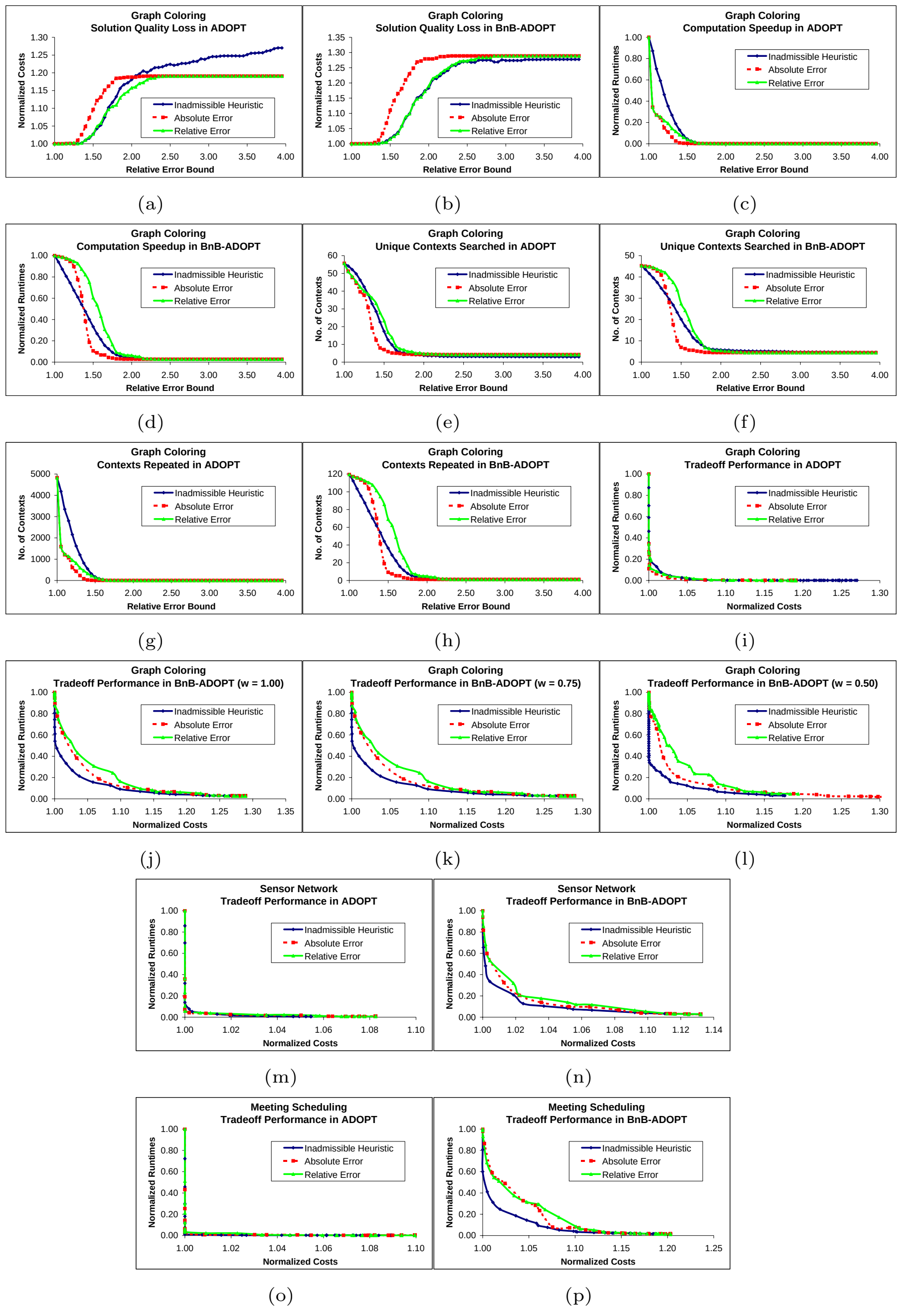


Fig. 6. Experimental Results

participant who has another meeting during the same time slot is infinity (to be precise: 1,000,000) since the same person cannot attend more than one meeting at a time. The cost of a non-scheduled meeting is 100. All other costs are randomly generated from 0 to 100. We average the experimental results over 50 DCOP instances.

We measure the runtimes, solution costs, and the number of unique (= different) and repeated contexts per variable. We measure the runtimes in cycles [8] and normalize them by dividing them by the runtimes of finding the minimal solution costs. We normalize the solution costs by dividing them by the minimal solution costs. We vary the relative error bounds (equal to the worst acceptable solution costs, normalized by dividing them by the minimal solution costs) from 1.0 to 4.0. We use the relative error bounds both as the relative error bounds for the Relative Error Mechanism and the multipliers for the admissible h-values for the Inadmissible Heuristics Mechanisms. We pre-calculate the minimal solution costs and use them to calculate the absolute error bounds for the Absolute Error Mechanism from the relative error bounds.

Figure 6 shows the results. In the following, we discuss only the results for graph coloring since the results for sensor network scheduling and meeting scheduling are similar:

- Figures 6(a,b) show that the normalized cost increases as the relative error bound increases, indicating that the solution quality of ADOPT and BnB-ADOPT decreases. The solution quality remains significantly better than predicted by the error bounds. For example, the normalized cost is less than 1.3 (rather than 3) when the relative error bound is 3.
- Figures 6(c,d) show that the normalized runtime decreases as the relative error bound increases, indicating that ADOPT and BnB-ADOPT terminate faster. Their normalized runtime decreases to almost zero when the relative error bound is about 1.5 for ADOPT and 2.0 for BnB-ADOPT.
- Figures 6(e,f,g,h) give insight into the reasons for the decrease in normalized runtime. They show that the number of unique and repeated contexts per variable decreases as the relative error bound increases, indicating that both the size of the search space explored by ADOPT and BnB-ADOPT and the size of the search space that they need to reconstruct decreases. The size of the search space that ADOPT needs to reconstruct is much larger than the one that BnB-ADOPT needs to reconstruct since ADOPT needs to reconstruct abandoned partial solutions repeatedly.
- Figures 6(i,j) give insight into the different mechanisms by plotting the normalized runtime needed to achieve a given normalized cost. For ADOPT, there does not seem to be a significant difference between the three mechanisms. However, for BnB-ADOPT, the Inadmissible Heuristics Mechanism performs better than the Absolute Error Mechanism, which in turn performs better than the Relative Error Mechanism. For example, the normalized runtime needed to achieve a normalized cost of 1.05 is about 0.18 for the Inadmissible Heuristics Mechanism, 0.30 for the Absolute Error Mecha-

nism and 0.35 for the Relative Error Mechanism. Thus, one should use the Inadmissible Heuristics Mechanism for BnB-ADOPT.

- Figures 6(k,l) give insight into the robustness of our conclusions for h-values of different quality. We use the h-values $w \times h$ for $w = 0.5$ and $w = 0.75$, where h denotes the DP2 h-values used before. The resulting graphs are similar to the graph from Figure 6(j) for $w = 1.0$. Thus, our conclusions do not significantly depend on the quality of the h-values. Note, however, that the result would have been different if we had used zero (= completely uninformed) h-values since then the Inadmissible Heuristics Mechanism does not apply since multiples of zero are equal to zero. However, such situations are rare.

8 Conclusions

We investigated three mechanisms that trade off the solution costs of two DCOP search algorithms (ADOPT and BnB-ADOPT) for smaller runtimes, namely the existing Absolute Error Mechanism, the new Relative Error Mechanism and the new Inadmissible Heuristics Mechanism. The solution costs that result from the two new mechanisms are bounded by a more meaningful quantity than the solution costs of the existing mechanism, since they both result in solution costs that are larger than minimal by at most a user-specified percentage. Furthermore, the Inadmissible Heuristics Mechanism dominates both the Absolute Error Mechanism and the Relative Error Mechanism for BnB-ADOPT. In general, we expect our mechanisms to apply to other DCOP algorithms as well since all of them perform search and thus benefit from using h-values to focus their searches.

Acknowledgment

This research has been partly supported by an NSF award to Sven Koenig under contract IIS-0350584. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the sponsoring organizations, agencies, companies or the U.S. government.

References

1. S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proceedings of AAMAS*, pages 1041–1048, 2005.
2. A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of AAMAS*, pages 1427–1429, 2006.
3. A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraints optimization. In *Proceedings of ECAI*, pages 103–107, 2006.
4. P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC4(2):100–107, 1968.

5. R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of AAMAS*, pages 310–317, 2004.
6. R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of AAMAS*, pages 438–445, 2004.
7. R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of IJCAI*, pages 224–229, 2005.
8. P. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.
9. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of IJCAI*, pages 1413–1420, 2005.
10. I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.
11. W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the Distributed Constraint Reasoning Workshop*, 2007.