Algorithm Selection for Constraint Optimization Domains

Avi Rosenfeld
Department of Industrial Engineering
Jerusalem College of Technology, Jerusalem, Israel 9116001
Email: rosenfa@jct.ac.il

***Abstract*—In this paper we investigate methods for selecting the best algorithms in classic distributed constraint optimization problems. While these are NP-complete problems, many heuristics have nonetheless been proposed. We found that the best method to use can change radically based on the specifics of a given problem instance. Thus, dynamic methods are needed that can choose the best approach for a given problem. We found that large differences typically exist in the expected utility between algorithms, allowing for a clear policy. We present a dynamic algorithm selection approach based on this realization. As support for this approach, we describe the results from thousands of trials from Distributed Constraint Optimization problems that demonstrates the strong statistical improvement of this dynamic approach over the static methods they are based on.**

I. Introduction

When multiple agents operate within a joint environment, inter-agent constraints typically exist between group members. Assuming these agents operate within a cooperative environment, the team must decide how to coordinate satisfying as many of these constraints as possible [21]. Instances of such problems are classic distributed planning and scheduling domains including specific applications such as supply chain management, disaster rescue management, Personal Data Assistant (PDA) scheduling, and military conflict planning [9], [19]. However, solving these real-world problems are challenging as they are known to be of NP-complete, or worse, complexities [10], [12], [19].

Despite the computational complexity inherent in these problems, a variety of algorithms have been suggested [4], [10], [11], [12], [15], [17], [21]. These algorithms differ in what and how agents communicate to attempt to find an optimal assignment. Each of these approaches have different resource cost requirements (e.g., time, number of messages), and are often useful in different problem classes. Thus, an important task for designers of these planning and scheduling systems is to find the algorithm that will work best for a given problem instance.

In this paper we claim that an algorithm selection approach is helpful in dictating which type of approach to use. The key to this approach is that differences between algorithms are typically quite large, and can be locally measured. This allows agents to locally control what information to transfer to group members. To demonstrate the effectiveness of this approach we study a general Distributed Constraint Optimization Problem (DCOP) domain [10], [11], [21]. We performed thousands of trials involving a variety team sizes and problem parameters and found that the described algorithm selection approach was effective in significantly outperforming the static methods they were based on.

II. Domain Formalization and Algorithms Description

In this section, we formally present a general Distributed Constraint Satisfaction and Optimization Problem domain (DCSP and DCOP respectively). The goal within a DCSP or DCOP problem is for distributed agents, each with control of some variables, to either satisfy (in DCSP) or to optimize (in DCOP) a global utility function. DCOP is a generalization of the DCSP problem as the goal is to minimize the number of non-fulfilled constraints, and is thus more suitable for most real-world problems [9]. The DCOP problem has been previously defined as follows [4], [10]:

- A set of N agents A = A_1, A_2 ... , A_N
- A set of n variables V = x_1, x_2 ... , x_n
- A set of domains D = D_1, D_2 ... , D_n where the value of x_i is taken from D_i. Each D_i is assumed finite and discrete.
- A set of cost function f = f_1, f_2 ... , f_m where each f_i is a function f_i: $D_{i,1} \times \ldots \times D_{i,j} \rightarrow$ N $\cup \infty$. Cost functions represent what must be optimized and are typically referred to as *constraints*.

- A distribution mapping Q : V $\rightarrow$ A assigning each variable to an agent. Q(x_i) = A_i denotes that A_i is responsible for choosing a value for x_i. A_i is given knowledge of x_i, D_i and all f_i involving x_i.
- An objective function F defined as an aggregation over the set of cost functions. Summation is typically used.

DCOP problems are often represented as connected graphs where nodes must be assigned one of k colors. For simplicity, the assumption is typically made that one assigns an agent to every node within the graph to decide how these nodes should be colored. Thus, the notation A_i and x_i can be used interchangeably [4]. In the DCSP variation the goal of the problems is for every node be assigned a color such that no connected node (often referred to as a neighbor within the graph) has the same color. There is a cost function of ∞ for having two connected nodes with the same color, or in other words, the DCSP contains a hard constraint between nodes (agents). The DCOP problem is a relaxation of this problem. Here the group's utility is based on minimizing the number of constraints that have not been satisfied. According to the DCOP formalization this referred to as F [11].

A range of algorithms exist for solving DCSP and DCOP problems. Well-known algorithms include distributed breakout (DBO) [21], asynchronous backtracking (ABT) [20], asynchronous weak-commitment (AWC) [20] and the Optimal Asynchronous Partial Overlay (OptAPO) [10]. In general, the DBO, ABT, and AWC algorithms are fully distributed algorithms. As such, each algorithm focuses on finding a solution without sending constraint information beyond the local agents (nodes in the graph) with which they have direct communication. These algorithms differ in what local information should be communicated between neighboring nodes, and how neighboring nodes should be prioritized to first attempt a solution. In contrast, the OptAPO requires merging semi-centralized solutions. This algorithm has a "mediator" stage where agents are allowed to directly communicate constraint information of non-local agents. This mediator agent can recommend a potential solution to a set of agents for which it mediates[1], allowing for a solution to be found much more quickly [10]. Thus, these algorithms not only differ in what constrain information is communicated, but also as the degree of problem centralization is used [4].

As the degree of centralization between OptAPO and other DCOP algorithms differs, debate exists how performance should be measured. The most common performance measure, which we based our experiments on, is how many cycles were needed to solve a given problem instance [20]. Within this measure, one unit of "time" is taken as the series of actions where agents process all incoming messages, process those messages, and send a response. In our experiments, we chose the more accepted cycle based measure to evaluation performance[2].

For example, Figure 1 provides an example of a simple DSCP (or DCOP) problem with 6 agents (nodes) and binary constraints (either black or white colors). At left, one finds the original problem state, and the right side provides a solution (or in the DCOP variation F = 0, or no constraints are broken).

We expected that different algorithms perform best in different problem and domain attributes. These attributes can include factors relating to the structure of the problem instance such as the number of nodes (agents), the total number of allowed node colors, and the density of connections between nodes (forming the problem constraints). Additionally, attributes that are external to the graph structure but are domain factors are also likely to be important in deciding which algorithm to use. These factors include the cost of communication between agents, if non-local communication is allowed and if so at what cost, and the time to find a solution.

[1]Note that this set of agents is typically of some size between the number of local agents and the total number of agents [4]. Thus, while OptAPO does not constitute a distributed, local solution, it does not constitute a classic centralized solution either.

[2]Other measures besides cycles have been proposed to evaluate the DCOPs' performance. Meisel et al. [7] have argued that performance should be measured based in terms of the number of computations each distributed agent performs and proposed a concurrent constraint checks measure (ccc) to quantify this amount. A hybrid measure, proposed by Davin and Modi [4], suggest using a Cycle-Based Runtime (CBR) measure that is parameterized between latency between cycles and computation speed as measured by the concurrent constraint checks measure. In their opinion, this measure between accounts for differences in centralization in DCOP algorithms. Note that if one chooses the ccc or CBR measures, or if new DCOP algorithms are found with better performance within the cycle based measure, they can be substituted to study the relative performance of the algorithms under consideration.

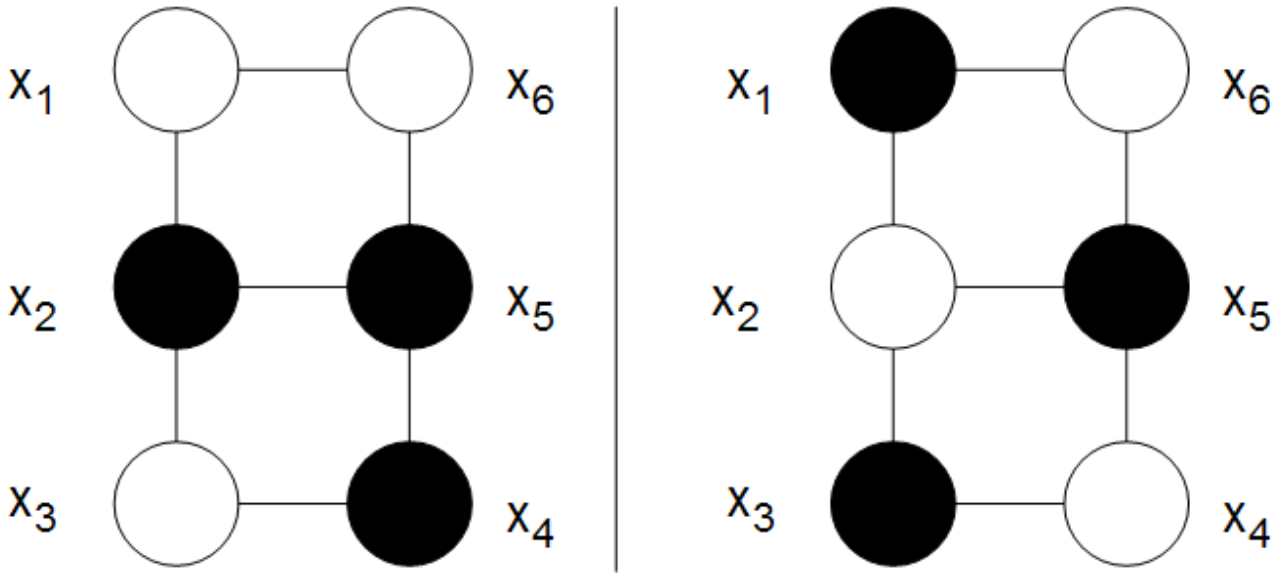

Fig. 1. A sample DCSP problem with 6 nodes (N=6) and 2 colors. The original problem state is at left, with one possible solution at right.

III. Using Phase Transitions to aid Algorithm Selection

This paper focuses on developing an algorithm selection approach for constraint optimization problems. Previously, Rice generally defined the algorithm selection problem as the process of choosing the best algorithm for any given instance of a problem from a given set of potential algorithms [16]. We define the constraint optimization coordination selection process used in this paper as follows: Let $GR = \{a_1, \ldots, a_N\}$ be a group on N agents engaged in some cooperative behavior. Each agent, a, can choose out of a library of coordination algorithms, $\{CA_1 \ldots CA_k\}$. We denote this selection CA_{a_j}, where $1 \leq a_j \leq k$.

Using CA_{a_j} affects the group's utility by a certain value, $\mathcal{UT}^a(CA_{a_j})$. $\mathcal{UT}^a(CA_{a_j})$ is composed of a gain, $\mathcal{G}(CA_{a_j})$, that the group will realize by using algorithm CA_{a_j}, and the agent's cost, $\mathcal{C}(CA_{a_j})$, by using that same algorithm. As this paper assumes the agents within these problems are cooperative, the goal is to maximize $\sum_{a=1}^{N} \mathcal{UT}^a(CA_{a_j})$. To achieve this, each agent must select the algorithm CA_{a_j} from the k possible algorithms in the library whose value $\mathcal{G}(CA_{a_j})$ - $\mathcal{C}(CA_{a_j})$ is highest.

One possible solution involves performing no learning in advance and instead attempts to identify the best algorithm exclusively during run-time. For example, Allen and Minton [1] suggest running all algorithms $\{CA_1 \ldots CA_k\}$ in the portfolio for a short period of time on the specific problem instance. Secondary performance characteristics are then compiled from this preliminary trial to select the best algorithm. Gomes and Selman [5] suggest running several algorithms (or randomized instances of the same algorithm) in parallel creating an algorithm portfolio. However, assuming running each algorithm incurs costs $\mathcal{C}(CA_1) \ldots \mathcal{C}(CA_k)$, these approaches are likely to be inefficient as the cumulative costs of running all algorithms are likely to be higher than the potential gain of finding even the optimal choice.

Instead, we claim that the best algorithm within these problems can be quickly identified based on finding phase transitions within these types of problem instances. The basis of this claim is the previous findings that NP-complete problems are not all equally difficult to solve [3], [13]. Many instances of NP-complete problems can still be quickly solved, while other similar instances of problems from the same domain cannot. They found that phase transitions are a well known phenomenon across which problems display dramatic changes in the computational difficulty and solution character [13].

The concept of phase transitions has been applied to differentiate classes of these "easy" and "hard" NP-complete problem instances [13]. Within distributed constraint satisfaction problems (DCSP),

these problems can typically be broken into an easy-hard-easy pattern [11], [13]. The first set of easy problems represent a category of under-constrained problems. All DCSP algorithms typically find an optimal solution quickly for these instances. At the other extreme, the second easy category of problems are those that are over-constrained. Within these problems, the same algorithms can typically demonstrate that no solution exist, and thus these algorithms end in failure. The hardest DCSP problems to solve are those within the phase transition going from under to over-constrained problems, a category of problems also called "critically constrained". These problems are the hardest to solve, with no solution often being found [13].

One may view the DCOP problem as a generalization of the more basic DCSP decision form of the problem. Again, in problems where there are few cost constraints, the optimization requirements are low, and an optimal solution can be quickly found. The "hard" problems exist where optimization requirements are high. However, debate exists if a third set of problems exist similar to the third "easy" problem set where DSCP problems can be quickly shown to have no solution. It would seem that even over-constrained DCOP instances cannot be easily optimized and still comprise "hard" problems [22]. Consequently, DCOP problems should be divided only into Easy-Hard categories (instead of Easy-Hard-Easy) or those easy problems to solve before the problem's phase transition, and "hard" problems after this point [14], [22]. Others have claimed [14] that certain optimization problems may in fact follow an easy-hard-easy distribution. However, this debate is not central to our thesis. According to both opinions, problem clusters do exist, and the difference of opinion revolves around the number of these clusters. If we follow the easy-hard model we should expect to see two clusters of problems with a transitionary phase between the two, but following the easy-hard-easy model should yield three such clusters with two transitionary phases.

Despite the computation complexity in solving all but trivial DCOP problems, a variety of algorithms can be used for attempting a solution in this domain. These algorithms impact when constraints are communicated between agents, thus impacting how the agents attempt to minimize F. We can formally expand the classic DCOP model into an algorithm selection based model by modeling the selection of algorithms $\{CA_1 \dots CA_j\}$ that each agent can choose in deciding what and how to communicate while attempting a solution. The intrinsically different approaches used by algorithms $\{CA_1 \dots CA_j\}$ makes them best suited for problems of differing levels of complexity.

This realization significantly simplifies the process of finding those problems instances where a given DCOP algorithm, CA_{a_j}, will be superior to other algorithms within $\{CA_1 \dots CA_j\}$. Based on this knowledge, we expect to find attributes that separate between fundamentally different types of problems. Assuming each algorithm is best suited for different clusters of problems, a clear policy will be evident as to which algorithm to select, even when agents are confined to using only locally available information. Instead of viewing all domain problems as an enormous state space where we must map the relative effectiveness of algorithms $\{CA_1 \dots CA_j\}$, we instead focus on finding the problem attributes that differentiate these algorithms, significantly reducing the state space. After these attributes have been found, we expect to be able to further cluster problems as "easy" or "hard" types of interactions. One type of algorithm will then be dominant within the easy problems, followed by phase shift(s)[3] where differences between algorithms are smaller and less apparent, followed by another large problem cluster where a second algorithm becomes dominant. As a result, our research focuses on two important questions: 1. What are the attributes that differentiate between algorithms $\{CA_1 \dots CA_j\}$? 2. At what attribute values should one switch between algorithms?

IV. Results

Our first step was to implement the algorithm library of the ABT, AWC, DBO, and OptAPO within the previously defined DCOP domain. To do this, we used the Farm simulation environment [6] to create randomized instances of 3-color optimization

[3]We refer to differences in problem instance clusters as phase shifts instead of phase transitions. This follows the distinction made by Brueckner and Parunak [2] who reserve the term "phase transition" to clusters that have been analytically derived and refer to "phase shifts" to describe problem clusters that have been emperically found. As this work derives these problem sets based on emperical observation, we use the second term.

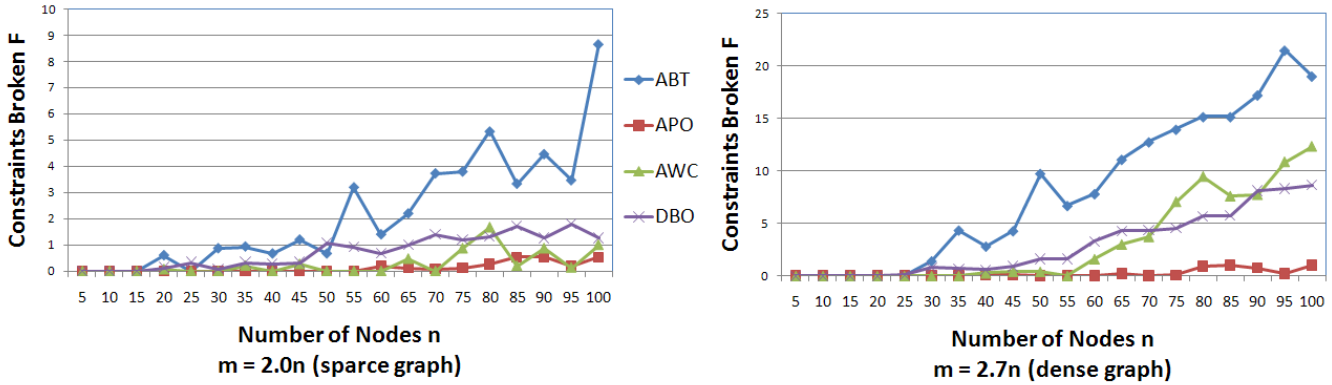

Fig. 2. Graph coloring performance with ABT, AWC, DBO, and OptAPO algorithms with random graphs with 5-100 nodes (X-axis) and edges = 2.0n (left) and 2.7n (right). Each datapoint represents averaged results from 30 runs with 100 cycles per run.

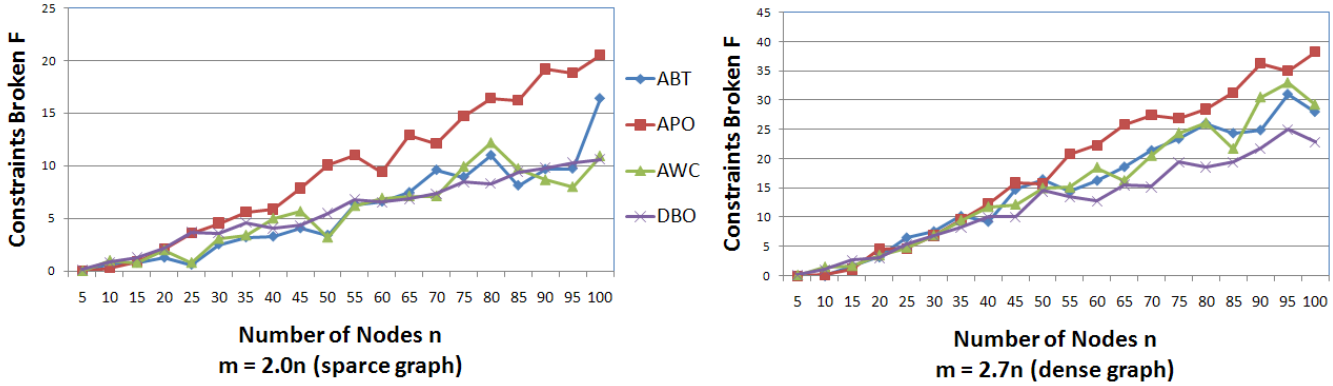

Fig. 3. Graph coloring performance with ABT, AWC, DBO, and OptAPO algorithms with random graphs with 5-100 nodes (X-axis) and edges = 2.0n (left) and 2.7n (right). Each datapoint represents averaged results from 30 runs with 10 cycles per run.

problems. We first varied parameters such as the number of nodes (agents) and edges (which control the number of constraints) within these problems. Specifically, we studied "sparse" coloring graph problems where the number of edges (m), is two times the number of nodes (n), and "dense" graph problems with 2.7 times the number of edges (m) to nodes (n). Traditionally, these parameters were thought to control if a problem instance would be "easy" or "hard" [10].

Figure 2 represents the performance results of the ABT, AWC, DBO, and OptAPO within problem instances. We measured the number of non-fulfilled constraints (F) after 100 cycles. The agents using each of the algorithms did not know in advance how much time would be available to reach a solution. As a result, after each cycle, the system would check if the global utility (F) had improved. If it had, it created a snapshot of this solution, so that the process could be interrupted at any time, and still return the best solution yet found. This allows for creating an interruptible anytime version of these algorithms as per previous definitions of anytime algorithms [23]. Note that in the easiest problems (with 30 or less nodes in both problem sets), no significant differences existed between algorithms as all algorithms were able to solve these problems equally (with the notable exception of ABT). Beyond this point, the OptAPO algorithm on average outperformed all other algorithms. This result is consistent with previous finding demonstrating the effectiveness of the OptAPO algorithm in solving challenging DCOP problems regardless of the number of nodes or constraints (edges) within the problem [10].

However, we found that the best algorithm to use

also differed radically based on parameters such as the time allotted to solve a problem instance. In Figure 3 we again ran the ABT, AWC, DBO, and OptAPO algorithms but allotted only 10 cycles of runtime. Note how the APO algorithm performs significantly worse than the other algorithms (in problems with > 40 nodes) with the DBO algorithm performing significantly better, especially in dense graphs with more than 60 nodes. This result is not surprising as the OptAPO is fundamentally different from other algorithms in the amount of problem centralization used. Evidently, this algorithms needs an initialization period in order that its "mediator" nodes have enough information about non-local nodes in order to attempt an effective solution. As such, in these cases this algorithm underperformed that of other purely localized algorithms that had no such overhead.

Communication costs can also radically affect which algorithm we should select. Recall that the goal of our algorithm selection model is to maximize $\sum_{a=1}^{N} \mathcal{UT}^a(CA_{a_j})$ where $\mathcal{UT}^a(CA_{a_j})$ is the gain $\mathcal{G}(CA_{a_j})$ the group achieves by using that algorithm minus the cost, $\mathcal{C}(CA_{a_j})$, paid by using the same algorithm. Again, the OptAPO algorithm is fundamentally different from other algorithms in that it uses non-local communication, giving this algorithm a potential cost $\mathcal{C}(CA_{a_j})$ not existent in other algorithms. Assuming such cost is significant – say because of privacy concerns or communication link cost, the OptAPO algorithm should also be avoided even if unlimited time exist to solve these problems. Indeed we found that the best of the breed of the localized algorithms, (such as the DBO or AWC algorithms), clearly outperforms OptAPO in these types of problem instances.

Figure 4 demonstrates the impact of non-local communication cost on algorithm selection. In this graph we compared the performance of the OptAPO and DBO algorithms in dense graph problems with 100 cycles allotted. When communication was free the APO algorithm (APO-100 Cost 0) did significantly outperform DBO. However, once non-local communication had a cost of 0.02 quality units per communication link, the DBO algorithm outperform OptAPO (APO-100 Cost 0.2)

Because of the radically different performance of these algorithms, the selection policy is often

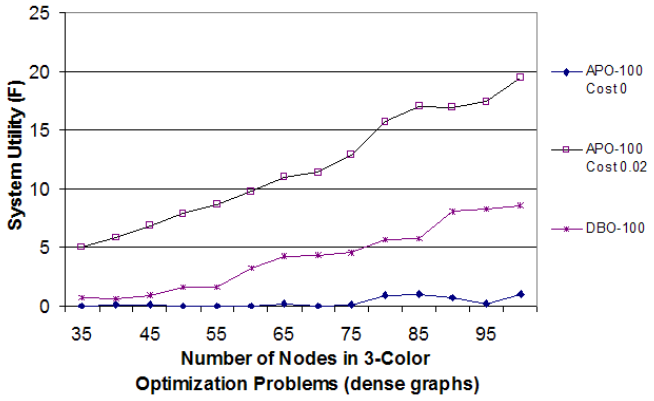

Fig. 4. The impact of non-local communication cost on algorithms studied

quite clear. Let us assume that agents are aware of performance limitations such as the time to complete the task, or the cost of non-local links. A clear policy typically becomes immediately evident. For example, assume there is no communication cost and agents need to find the best DCOP solution given a relative long period of time (e.g. 100 cycles or more). OptAPO is then clearly the best choice. Conversely, assuming communication is costly (e.g. cost of 0.02 or more), or only a very short period of time is allocated (e.g. time of 10 cycles or less), the best of the local algorithms, e.g. DBO, was selected. In cases with problem attributes between those with a clearly defined policy (e.g. 50 cycles of time to solve the problem), we considered two possibilities. In the first possibility, a random selection is taken between the borderline algorithms. A second possibility is to calculate the midpoint within the attribute space between these algorithms and to choose the first algorithm for instances before the midpoint and use the second algorithm after this point. While random selection or midpoint heuristics will not likely form the optimal choice in many of these instances, we hypothesized the difference between algorithms in these cases is not large as this the transitional range for this attribute. Thus, the difference between optimal and non-optimal choices within these types of problems was not expected to deviate significantly from the optional choice.

Figure 5 demonstrates the effectiveness of the *Selection* algorithm just described. For comparison, we also display the average group utility (F) as taken from the the static DBO and OptAPO algorithms. We also created an *Optimal* group could run all

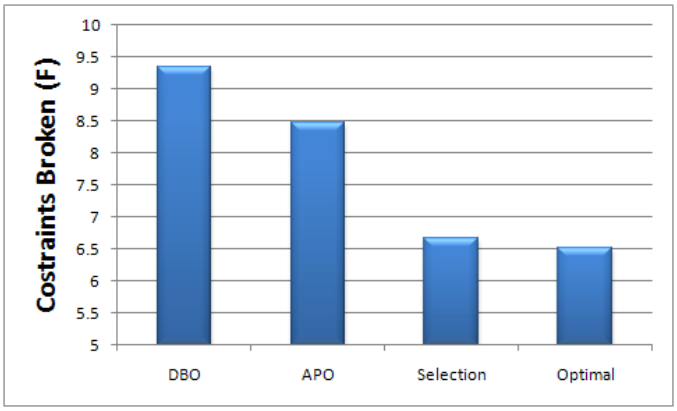

Fig. 5. Comparing the effectiveness of the *Selection* algorithm policy versus the static DBO and OptAPO algorithms

static algorithms without cost, and then accept the algorithm that returned the highest utility. The results in Figure 5 were generated from a total of 100 DCOP 3-color graph problems with random problem attributes in the time to solve the problem, the number of nodes and edge constraints, and non-local communication costs. As these problems instances were randomly selected, many of these problems had problem attributes fell within the problem space where a clear policy existed. Such instances included: instances with long times to solve the problem, very short times, or where non-local communication had a significant cost. In order to strengthen the significance of this experiment, we ensured that at least 25 percent of the problem instances were taken from the category when no clear policy existed. Notice that the *Selection* approach closely approximated the optimal choice, and significantly outperformed statically choosing either DBO or OptAPO. In order to evaluate the statistical significance of these findings, we performed a two-tailed t-test to compare the *Selection* approach to the static DBO and OptAPO methods. The resulting p-score was well below 0.05 (0.02), supporting the significance of the presented approach. Similarly, we compared the dynamic approach with the optimal selection policy and found only an insignificant difference (p-score greater than 0.8) between these values. This supports the claim that randomly selecting between algorithms in borderline cases does not significantly hurt performance.

V. Conclusion and Future Work

In this work we present an algorithm selection approach for solving constraint optimization problems. We focused on two factors: what attributes differentiate between the algorithms and how can we build a selection policy based on those attributes. We present strong empirical evidence of the success of this approach in a general DCOP domain, suggesting the generality of this work.

For future work, several directions are possible. In this work, we manually found the attributes that differentiated the coordination algorithms within the domains we studied. We hope to study how algorithms can be created to automate this process so that novel interaction measures may be learned for quantifying coordination in other domains.

The success of our coordination selection approach was rooted in the realization that different clusters of problems can be created based on the hardness of different agent interactions. We drew upon the "phase transition" concept used to describe some constraint satisfaction problems [13]. However, following Brueckner and Parunak [2] we reserve the term "phase transition" to refer to a term used by physicists for mathematically describable behavior within the system, and instead term the clusters of problems we empirically observed as phase shifts. We hope to study in the future what formal models can be created that can predict where and when these transitions should occur. We believe this study could strengthen the theoretical basis of the work we present.

References

[1] John A. Allen and Steven Minton. Selecting the right heuristic algorithm: Runtime performance predictors. In *Canadian Conference on AI*, pages 41–53, 1996.

[2] S. Brueckner and H. Parunak. Information-driven phase changes in multi-agent coordination. 2003.

[3] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *IJCAI-91*, pages 331–337, 1991.

[4] John Davin and Pragnesh J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS '05*, pages 1057–1063, 2005.

[5] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence (AIJ)*, 126(1-2):43–62, 2001.

[6] Bryan Horling, Roger Mailler, and Victor Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.

[7] A. Meisels I. Razgon E. Kaplansky and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, July 2002.

[8] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

[9] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS '04*, pages 310–317, 2004.

[10] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04*, pages 438–445, 2004.

[11] Roger Mailler and Victor Lesser. Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. In *AAMAS '04*, pages 446–453, New York, 2004.

[12] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[13] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic "phase transitions". *Nature*, 400(6740):133–137, 1999.

[14] H. Van Dyke Parunak, Sven Brueckner, John Sauter, and Robert Savit. Effort profiles in multi-agent resource allocation. In *AAMAS '02*, pages 248–255, 2002.

[15] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

[16] J. R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 118–165, 1976.

[17] Evan Sultanik, Pragnesh Jay Modi, and William C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, pages 1531–1536, 2007.

[18] Evan Sultanik, Pragnesh Jay Modi, and William C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, pages 1531–1536, 2007.

[19] Willem Jan van Hoeve, Carla P. Gomes, Bart Selman, and Michele Lombardi. Optimal multi-agent scheduling with constraint programming. In *AAAI/IAAI*, pages 1813–1818, 2007.

[20] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

[21] Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In Victor Lesser, editor, *ICMAS*. MIT Press, 1995.

[22] Weixiong Zhang. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In *Principles and Practice of Constraint Programming*, pages 153–167, 2001.

[23] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.