Getting What You Pay For: Is Exploration in Distributed Hill Climbing Really Worth It?

Melanie Smith and Roger Mailler
Computational Neuroscience and Adaptive Systems Lab
University of Tulsa, Oklahoma, USA
http://www.cnas.utulsa.edu
{melanie,mailler}@utulsa.edu

***Abstract*—The Distributed Stochastic Algorithm (DSA), Distributed Breakout Algorithm (DBA), and variations such as Distributed Simulated Annealing (DSAN), MGM-1, and DisPeL, are distributed hill-climbing techniques for solving large Distributed Constraint Optimization Problems (DCOPs) such as distributed scheduling, resource allocation, and distributed route planning. Like their centralized counterparts, these algorithms employ escape techniques to avoid getting trapped in local minima during the search process. For example, the best known version of DSA, DSA-B, makes hill-climbing and lateral escape moves, moves that do not impact the solution quality, with a single probability p. DSAN uses a similar scheme, but also occasionally makes a move that leads to a worse solution in an effort to find a better overall solution. Although these escape moves tend to lead to a better solutions in the end, the cost of employing the various strategies is often not well understood. In this work, we investigate the costs and benefits of the various escape strategies by empirically evaluating each of these protocols in distributed graph coloring and sensor tracking domains. Through our testing, we discovered that by reducing or eliminating escape moves, the cost of using these algorithms decreases dramatically without significantly affecting solution quality.**

***Index Terms*—distributed constraint optimization, efficiency**

I. Introduction

Distributed hill-climbing algorithms are very powerful, practical tools for solving numerous real-world problems including distributed scheduling, resource allocation, and distributed route planning. These problems can be easily mapped to distributed constraint optimization, and like DCOPs, they must be solved using algorithms that can make decisions about how to best improve the global state of the problem from an agent's limited, local perspective. The ultimate goal of distributed constraint optimization is to find a optimal solution while minimizing the overall cost incurred to find it (i.e. the required bandwidth, processing cycles, messaging, etc.)

Complete algorithms, such as Adopt and its variants [**?**], Optimal Asynchronous Partial Overlay (OptAPO) [**?**], and DPOP [**?**], are guaranteed to find an optimal solution, but suffer because the cost they incur to find it limits their scalability. So, in practice, one must accept a close-enough solution, especially if the problem is large or the solution needs to be derived quickly. Hill-climbing or local search algorithms tend to work very quickly even on large problems, although they cannot guarantee the best solution and cannot identify when there is no solution for DCSP. This is because these algorithms are prone to getting trapped in local minima. Accordingly, various heuristics have been devised to escape local minima.

The Distributed Stochastic Algorithm (DSA) is a simple hill-climbing technique that works by having agents change their value with probability p when making that change will improve their solution. The setting of p can have dramatic effects on the behavior of the algorithm and can be quite problem specific. For instance, on very dense problems, having high values of p can cause the protocol to converge more quickly, but the same setting on a sparse problem will cause it to oscillate unnecessarily.

Like other local search techniques, DSA employs an escape strategy for situations where the agent cannot change its value to a better, less conflicted value. One particularly dominant strategy, seen in the DSA-B variant, allows an agent with a locally sub-optimal value that cannot improve the solution to switch to another equally conflicted value. In other words, it can move laterally in the solution space by changing its value, but the overall solution does not improve. In DSA-B, these lateral moves are chosen with the same probability p as hill-climbing moves. Variants such as DSA-B1 and Distributed Simulated Annealing have different escape strategies that allow the algorithm to get out of the minima by taking a lateral or a downhill (bad) move. In this paper, we introduce the concept of treating the lateral escape moves differently than the hill-climbing and downhill moves to examine the efficiency of escape strategies used in each family of algorithms.

The Distributed Breakout Algorithm (DBA) [**?**] is a distributed adaptation of the centralized breakout algorithm [**?**]. DBA works by alternating between the *wait_ok?* and *wait_improve* modes, where messages are passed between agents and new values are appropriately adopted. When an agent detects a quasi-local-minimum [**?**], the agent uses its escape stratgy that increases the weights on all of its suboptimal cost functions. The Maximum Gain Message (MGM) protocol [**?**] is a simplified version of the DBA protocol that

does not change constraint costs to break out of local minima, essentially removing the escape strategy.

Although escape moves often lead to a better solution in the end, the cost of employing these escape strategies is often not well understood. In this paper, we investigate the efficiency (cost:benefit ratio) of various escape strategies by empirically evaluating each of these protocols in two distributed domains. Our results show that by reducing or eliminating escape moves, the cost of using these algorithms decreases dramatically without significantly affecting solution quality.

II. Distributed Constraint Optimization

A Distributed Constraint Optimization Problem (DCOP) consists of the following [**?**]:

- a set of *n* variables $V = \{x_1, \ldots, x_n\}$.
- a set of g agents $A = \{a_1, \ldots, a_g\}$
- discrete, finite domains for each of the variables $D = \{D_1, \ldots, D_n\}$.
- a set of cost functions $f = \{f_1, \ldots, f_m\}$ where each $f_i(d_{i,1}, \ldots, d_{i,j})$ is function $f_i : D_{i,1} \times \cdots \times D_{i,j} \rightarrow N \cup \infty$.

The problem is to find an assignment $S^* = \{d_1, \ldots, d_n | d_i \in D_i\}$ such that the global cost, called F, is minimized. Although the algorithms presented will work for any associative, commutative, monotonic aggregation function defined over a totally ordered set of values, with min and max elements, in this paper, F is defined as follows: $F(S) = \sum_{i=1}^{m} f_i(S)$.

In DCOP using variable decomposition, each agent is assigned one or more variables along with cost function associated with those variables. The goal of each agent, from a local perspective, is to ensure that it gets the best possible solution for its variables. For each of the agents, achieving this goal is not independent of the goals of the other agents in the system. In fact, in all but the simplest cases, the goals of the agents are strongly interrelated.

In this paper, for the sake of clarity, each agent is assigned a single variable. Since each agent is assigned a single variable, the agent is referred to by the name of the variable it manages. Also, this paper considers only binary cost functions that are of the form $f_i(d_{i,1}, d_{i,2})$. It is fairly easy to extend all the algorithms presented in this paper to handle more general problems where these restrictions are removed, either by changing the algorithm or by changing the problem as done in [**?**].

Additionally, throughout this paper the word *neighbor* is used to refer to agents that have variables that are related by a cost function. In other words, if an agent A has a cost function f_i that contains a variable owned by some other agent B, then agent A and agent B are considered neighbors.

III. Distributed Stochastic Algorithm (DSA)

The Distributed Stochastic Algorithm (DSA) is one of a class of algorithms based on the idea that at each step, each variable should change to its best value with some probability $p \in [0, 1]$. Because each variable changes with p probability, the likelihood of two neighbors changing at the same time is p^2. As long as p is selected correctly, the protocol will hill climb to a better state.

Algorithm	$improve > 0$	$improve = 0$	
	conflict	conflict	no conflict
DSA-A	v with p	-	-
DSA-B	v with p	v with p	-
DSA-C	v with p	v with p	v with p
DSA-D	v	v with p	-
DSA-E	v	v with p	v with p

TABLE I
DSA Variants [**?**].

Algorithm 1: DSA Main Procedure

```
main
    while not terminated do
        update agent_view with incoming ok?(x_j, d_j) messages;
        new_value ← choose_best_value;
        if new_value ≠ d_i and random ≤ p then
            d_i ← new_value;
            send(ok?, (x_i, d_i)) to all x_j ∈ neighbors;
        end
    end
end
```

One of the greatest benefits of the DSA protocol is that it uses considerably fewer messages than the Distributed Breakout Algorithm [**?**] because agents communicate only when they change values. As the protocol executes and the number of violations decrease, so do the number of messages. Algorithm 1 sketches the DSA main procedure. First, the agent receives messages from other agents and stores them in its $agent\_view$. Then a new_value is selected that best minimizes the cost function, i.e. has the greatest amount of improve. The agent adopts the new value and sends messages to its neighbors if the new value is different than its current value and a randomly generated number ($random \in [0, 1]$) is less than probability p.

DSA has a number of implementation variants, as shown in Table II. Zhang [**?**], [**?**] experimented with changing an agent's value v with probability p at times when there is improve (an expected hill-climbing move) and when there is no improve with and without a conflict existing in the world (a lateral move). However, the same value of p was used for all cases. Their work focused on the effects of an agent having the chance to change in different variations of those situations, and found that one variant, DSA-B, was the best approach because it is able to escape certain types of local minima in the search space by making lateral moves.

The DSA protocol is quite popular because it is by far the easiest hill-climbing protocol to implement. However, it is also one of the hardest to tune because it requires the user to specify p. The process of choosing this value can require a great deal of empirical testing because it is problem specific. The result of Zhang's work indicated that the superior case for DSA is the DSA-B variant with $p = 0.3$ [**?**].

Algorithm	$improve > 0$ conflict	$improve = 0$ conflict	$improve = 0$ no conflict	$improve < 0$ conflict
DSA-B	v with p	v with p	-	-
DSA-BE	v with p_H	v with p_L	-	-
DSA-B1	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-B1E	gv with p_H/g, ov with p_2/o	gv with p_L/g, ov with p_2/o	ov with p_2/o	ov with p_2/o
DSAN	rv	rv	rv	rv with t_i
DSANE	rv	rv with p_L	rv with p_L	rv with t_i

TABLE II

DSA VARIANTS FOR EXPERIMENTATION. v IS A VALUE TAKEN WITH PROBABILITY p. gv IS ANY OF THE VALUES RESULTING IN THE BEST $improve$. ov IS ANY OF THE OTHER VALUES THAN THE CURRENT VALUE OR AN ELEMENT OF gv. g AND o ARE THE NUMBER OF gv AND ov VALUES, RESPECTIVELY. rv IS A RANDOM VALUE. p_1, p_2, p_L, p_H ARE THE RESPECTIVE PROBABILITIES FOR TAKING A GOOD MOVE, AN OTHER MOVE, A LATERAL MOVE, AND A HILL-CLIMBING MOVE. t_i IS A PROBABILITY BASED ON THE TEMPERATURE SCHEDULE FOR CYCLE i.

We extend the DSA-B variant to allow the agent to execute a lateral move with a different probability than it would a hill-climbing move, which forms the crux of how we evaluate each algorithm in this paper. We call this extended form DSA-BE (DSA-B Extended). In Algorithm 1, the conditional statement that determines whether d_i is assigned a new value will be amended to include how much of an improvement is possible should the agent take on the new value. As shown in Table II, if $improve > 0$ and $random < p_H$, then new_value is assigned to d_i. Otherwise, if $improve == 0$ and $random < p_L$, then new_value is assigned. Else, no change to d_i is made. If $p = p_L = p_H$, then DSA-B and DSA-BE are equivalent.

A. DSA-B1

DSA-B1 [**?**] is another variant of DSA-B that takes good moves ($improve \geq 0$) with probability p_1/g and then makes an escape move with probability p_2/o, as detailed in Table II. The escape moves can occur no matter what the improve value is. The main procedure for DSA-B is adapted to include each of these conditions, similarly to how DSA-BE is described.

When the lateral and hill-climbing probabilities are incorporated into this variant (resulting in DSA-B1E), three probabilities are used: p_L and p_H when $improve = 0$ and $improve > 0$, respectively, and p_2 when making a downhill move for any improve value. Setting $p_1 = p_L = p_H$ is equivalent to DSA-B1.

B. Distributed Simulated Annealing (DSAN)

Distributed Simulated Annealing [**?**] is a variation on DSA where random moves are taken with an exponentially decreasing schedule of temperatures, $T = \{t_1 \dots t_f\}$ when there is no improve ($improve \leq 0$) or a probability of 1 if $improve \geq 0$ (see Algorithm 2). We have chosen a schedule of temperatures that correspond to the number of cycles that have elapsed, following what was presented by Arshad and Silaghi [**?**]: $t_i = e^{(improve/maxImprove)/(numIterations/i^2)}$, where $numIterations$ is the number of cycles being executed and i is the current iteration number. The $maxImprove$ is the difference in the cost of the best and worst possible values. If $p_L = 1$ in DSANE, DSANE is equivalent to DSAN.

Our extended version of DSAN, DSANE, simply adjusts the probability of taking lateral moves to be equal to p_L instead of 1. This will add another condition to DSAN's Main Procedure, similar to the other adaptations we have made.

Algorithm 2: DSAN Main Procedure

```
main
    while not terminated do
        update agent_view with incoming ok?(x_j, d_j) messages;
        new_value ← choose_random_value;
        if new_value has same or fewer conflicts as d_i then
            d_i ← new_value;
            send(ok?, (x_i, d_i)) to all x_j ∈ neighbors;
        else
            if random < t_i then
                d_i ← new_value;
                send(ok?, (x_i, d_i)) to all x_j ∈ neighbors;
            end
        end
    end
end
```

IV. DISTRIBUTED BREAKOUT ALGORITHM (DBA)

The Distributed Breakout Algorithm (DBA) [**?**] is a distributed adaptation of the Centralized Breakout Algorithm [**?**]. DBA works by alternating between two modes. In the first mode, called the *wait_ok?* mode, the agents wait until they have received an *ok?* messages from each of their neighbors. Each *ok?* message contains the value that is current assigned to a neighbor's variable and are used to calculate the current cost.

Once the agents have a clear picture of the state of their variable, they can calculate the best alternative value assignment and the associated improvement to the cost function. The agents then sends out an *improve?* message, which contains their improve value, to each of their neighbors and change to the *wait_improve?* mode.

In the *wait_improve* mode, the agents collect *improve?* messages from each of their neighbors. Once all of the messages have been received, the agents check to see if their improvement is the best among their neighbors. If it is, it changes its value to the new improved value. If an agent believes it is in a QLM, it increases the weights on all of its suboptimal cost functions. Finally, the agents send *ok?* messages to each of their neighbors and change back to the *wait_ok?* mode. The algorithm starts up with each agent sending *ok?* messages and going into the *wait_ok?* mode.

Because of the strict locking mechanism employed in the algorithm, the overall behavior of the agents is to simultaneously switch back and forth between the two modes. So, if one or more of the agents reacts slowly or messages are delayed, the neighboring agents wait for the correct message to arrive. This makes the protocol's communication usage very predictable because in each mode, each agent sends exactly one message to each of its neighbors. Thus, if there are m constraints, exactly $2m$ messages are transmitted during each step.

The locking mechanism in DBA can be very beneficial because it does not allow neighboring agents to change their values at the same time, which prevents oscillations. However, it can also prevent opportunities for additional parallelism because it limits the number of variables that can change at each *wait_improve* step to at most half when they are in a fully connected problem. These limitations effectively allow at most 25% of the variables to change during any individual step of the protocol's execution.

A. Maximum Gain Message (MGM)

Another similar algorithm, MGM-1 [**?**], is described as strictly DBA with no escape strategy. This means that nothing will be done if a QLM is reached since there is no detection of whether the algorithm has landed in a minima. MGM-2 is a variation that allows randomly selected agents to act as offerers and propose a coordinated move with one of its neighbors. Both of these algorithms monotonically improve the solution, which is often a desirable feature when the penalty for a failed coordination is high. One interesting feature of these algorithms is that we can calculate a bound on their optimality using k-optimality [**?**]. A k-optimal solution is one that cannot be improved without the coordinated action of k or more agents. The DSA, DBA, and MGM protocols are 1-optimal, MGM-2 is 2-optimal, and complete algorithms are n-optimal[**?**].

B. DisPeL

The Distributed, Penalty-driven Local search algorithm (DisPeL) [**?**] is a synchronous iterative improvement protocol, and like DBA, agents send messages to each of their neighbors on every cycle. However, the difference is that DisPeL associates penalties to domain values rather than to constraints. There is a comparison between DBA and DisPeL in [**?**], showing that DisPeL outperformed DBA solving more problems at less median time on a car sequencing and graph-coloring domain. However, there was no mention of communication requirements.

V. Experimentation

To test our DSA and DBA variants, we implemented each in a distributed 3-coloring domain. The test cases were compared on two main factors to determine the variant's efficiency - total messages received, number of conflicts, and improvement per message. During each time cycle, the current cost and the number of messages transmitted were measured. These values were used to plot the graphs shown throughout this paper.

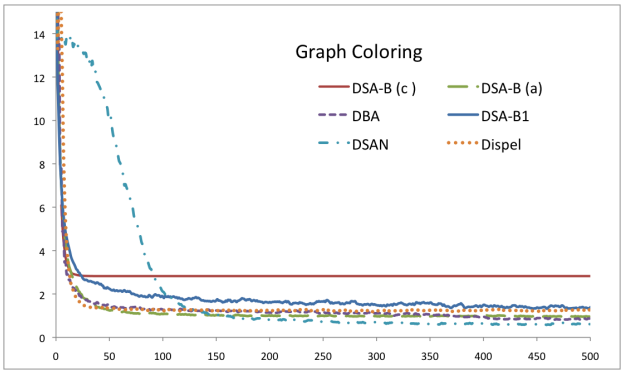

Fig. 1. Remaining Conflicts over Time for selected graph coloring agents with 300 nodes at 2.3n edge density.

A. Graph-Coloring Domain

Following directly from the definition for a DCOP, a graph coloring problem, also known as a k-colorability problem, consists of the following:

- a set of *n* variables $V = \{x_1, \ldots, x_n\}$.
- a set of *g* agents $A = \{a_1, \ldots, a_g\}$.
- a set of possible colors for each of the variables $D = \{D_1, \ldots, D_n\}$ where each D_i has exactly k allowable colors.
- a set of cost functions $F = \{f_1, \ldots, f_m\}$ where each $f_i(d_i, d_j)$ is function that returns 1 if $d_i = d_j$ and 0 otherwise.

The problem is to find an assignment $S^* = \{d_1, \ldots, d_n | d_i \in D_i\}$ such that $F(S) = \sum_{i=1}^{m} f_i(S)$ is minimized. Like the general DCOP, graph coloring has been shown to be NP-complete for all values of $k > 2$.

B. Setup and Results

For this domain, we conducted experiments that have *n* variables and *m* binary constraints. The test series consisted of random graphs with $n = \{100, 200, 300, 400, 500\}$ variables and $m = \{2.0n, 2.3n, 2.7n\}$ constraint densities to cover under-constrained, within the phase transition, and over-constrained environments. For each configuration of n and m, 10 graphs were created and 3 different cases (initial colorings) of each graph were run, giving 30 problems that were solved. The random seeds for creating the graph layouts and initial colorings were saved, so each variant solved the same set of problems. Each run was given 500 cycles of execution time.

During a cycle, each agent was given the opportunity to process its incoming messages, change its value, and queue up messages for delivery during the next cycle. The actual amount of execution time per cycle varied depending on the cycle, the problem, and the probability values. Running this set of 4500 tests took approximately 1500 hours of execution time. Due to space constraints, we show graphical results for within the phase transition (2.3n) using 300 nodes unless noted otherwise. Table III shows the complete results for all densities tested with 300 nodes.

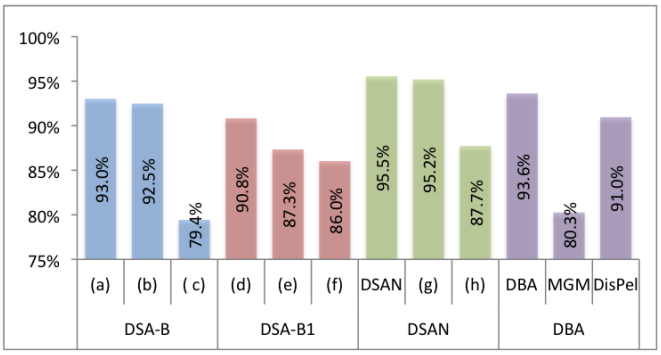

Fig. 2. Percentage of Conflicts Resolved per algorithm.

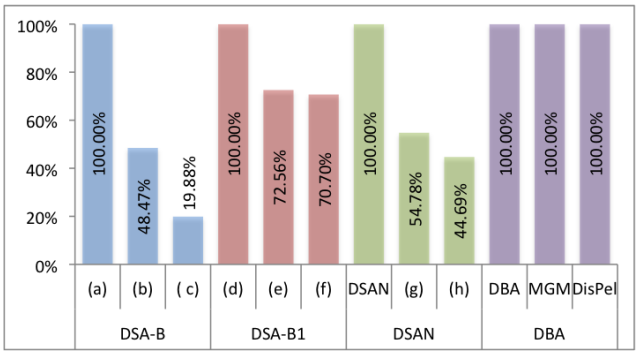

Fig. 3. Groupwise Comparison of Total Messages Received per algorithm.

We ran our tests in four groups: DSA-B, DSA-B1, DSAN, and DBA. Each of these groups has three configurations that were tested. As the control, we implemented the standard version of the algorithm. The other two configurations are the versions of the algorithm that incorporate a separate probability, p_L, that is used to differentiate lateral escape moves from the other types of moves. The particular probability values used for each configuration were set to the best values as determined by the algorithm's creators for the standard version, and the others give the test case for $p_L = 0$ and then a small value of p_L is used to give a limited escape strategy. The numerical values are included in Table III.

C. Metrics

Final solution quality is the most obvious metric to compare the efficiency of different algorithms. However, with hill-climbing approaches, the end solution is important, but what sets them apart is their overall efficiency in arriving at an acceptable solution.

Figure 1 shows several of the agents we tested and how the number of conflicts remaining changes as time passes. The solution quality at the end of 500 cycles differs by about 1 conflict, indicating that all of the algorithms produce competitive solutions. In fact, differences are so slight that they are not statistically different, and all were well within a standard deviation of one another.

However, even with such a negligible difference in end solution quality, these algorithms should not be considered equivalently efficient because they converge on their final solution with considerably different costs. Thus, instead of relying solely on solution quality, we use a metric to measure the ratio between cost and benefit by calculating the conflicts resolved per message ($efficiency = (RES/TMR)$, where RES is the conflicts resolved and TMR is the total messages received. For the 300 node x 2.3n density case, $initConflicts = 229$, the average number of starting conflicts. In this way we consider both the quality of the solution being found as well as the overall expense for finding it.

D. Results

The raw numerical results are listed in Table III. We use percentages in this section to compare the escape strategies of

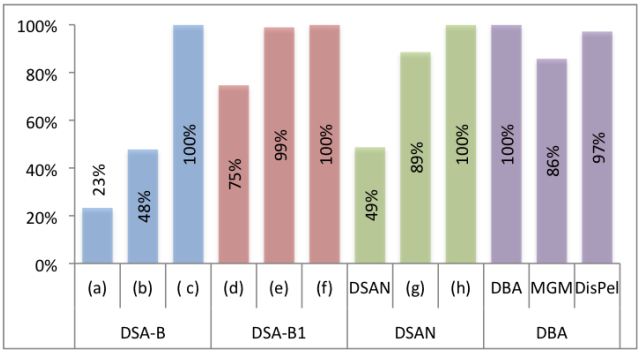

Fig. 4. Groupwise Comparison of Efficiency per algorithm.

each group. However, a side effect of that is not knowing how each algorithm compares to one another across the different groups. The raw numerical results are provided for this purpose.

Figure 2 shows the Conflicts Resolved by each algorithm as a percentage of the intial conflicts of the problem. In the majority of the cases, between 85% and 95% of conflicts were resolved after 500 iterations. We did not run the experiments longer than that as most of the algorithms had stopped making progress, which for a few of them (including DSAN and DisPeL) meant that they had not stopped progressing. However, one of the aims and benefits of developing hill-climbers is to arrive at a solution quickly and we do not know if a better solution would be found in a couple hundred, thousand, or more cycles. If time was not an issue, a complete algorithm would be used. One other question, however, is how much it costs to arrive at each solution?

Figure 3 shows a groupwise comparison of total messages received for each algorithm. In all groups but DBA, the lessening of lateral movement affected the number of messages dramatically. DSA-B(c), DSA-B1(f), and DSAN(h) all have no allowance for lateral movement. As you can see, the dropoff is considerable, with DSA-B(c) using a mere 20% of the messages DSA-B requires. Some escape moves are allowed by DSA-B1 and DSAN, which accounts for the higher amount of messages. DSAN reduces the probability of taking escape moves over time, so likely it sends more messages than DSA-B1 because the temperature starts off higher than DSA-

Algorithm	Configuration	$2.0n$ density RES	TMR	EFF	REM	$2.3n$ density RES	TMR	EFF	REM	$2.7n$ density RES	TMR	EFF	REM
DSA-B (a)	$p = 0.3$	210.70	7227	0.029	9.3	213	10391	0.020	16	243.20	13789	0.018	26.80
DSA-B (b)	$p_L = 0.1$, $p_H = 0.3$	210.80	3871	0.054	9.2	211.8	5037	0.042	17.2	240.90	6892	0.035	29.1
DSA-B (c)	$p_L = 0.0$, $p_H = 0.3$	188.30	1786	0.105	31.70	181.9	2066	0.088	47.1	209.10	2497	0.084	60.9
DSA-B1 (d)	$p_1 = 0.2$, $p_2 = 0.005$	210.00	6115	0.034	10	208	6487	0.032	21	231.00	8344	0.028	39
DSA-B1E (e)	$p_L, p_2 = 0.005$, $p_H = 0.2$	194.00	4010	0.048	26	200	4707	0.042	29	223.50	5657	0.040	46.5
DSA-B1E (f)	$p_L = 0.0$, $p_H = 0.2$	192.00	4001	0.048	28	197	4586	0.043	32	224.00	4932	0.045	46
DSAN	-	214.23	68044	0.003	5.77	218.8	73520	0.003	10.20	250.23	81918	0.003	19.77
DSANE (g)	$p_L = 0.2$	215.67	34513	0.006	4.33	218	40274	0.005	11	247.60	48678	0.005	22.4
DSANE (h)	$p_L = 0.0$	201.10	27047	0.007	18.9	200.9	32859	0.006	28.1	229.90	40978	0.006	40.10
DBA	-	217.63	598800	0.00036	2.37	214.4	688620	0.00031	14.6	240.50	808380	0.00030	29.5
MGM-1	-	188.70	598800	0.00032	31.3	183.8	688620	0.00027	45.2	206.30	808380	0.00026	63.7
DisPeL	-	204.50	598800	0.00034	15.5	208.3	688620	0.00030	20.7	236.10	808380	0.00029	33.9

TABLE III
RESULTS/CONFIGURATIONS: CONFLICTS RESOLVED (RES), TOTAL MESSAGES RECEIVED (TMR), EFFICIENCY (EFF), AND CONFLICTS REMAINING (REM) FOR 300 NODES WITH 2.0N, 2.3N, AND 2.7N DENSITIES IN THE GRAPH COLORING DOMAIN.

B1's probability for escape, even though it plumets quickly. DBA, MGM, and DisPeL, however, show no difference in the number of messages used, regardless of whether there is an escape strategy, because the algorithm always sends $2m$ messages during each cycle as a locking mechanism.

Figure 4 shows the groupwise comparison of efficiency over all the algorithm groups. In the DSA variants, we see that efficiency increases as escape moves decrease. The same algorithms that used the least amount of messages compared to their group also have the highest efficiency. It should also be noted that there is a big gap in efficiency scores for the DSAN and DBA groups compared to the DSA-B and DSA-B1 groups, and the raw efficiency score (see Table III) should be used to compare efficiency scores of algorithms in different groups.

Putting this all together, we find that DSA-B(c) uses 80% less messages as DSA-B(a) and results in approximately a 13% drop in solution quality. This means DSA-B(a) is 23% as efficient as DSA-B(c) that disallows lateral movement. Allowing some lateral movement, as in DSA-B(b), gives us just more than a 50% improvement in efficiency over traditional DSA-B(a) by using 48% as many messages without affecting solution quality by more than 0.5%. The DSA-B1 algorithms can tell a similar story except (e) and (f) are approximately the same in efficiency. This occurs because even though no lateral moves are being made, there is still a chance of making another move, which can facilitate moving out of a minima. The DSAN group has very similar percentages as the DSA-B results, because it cuts out all movement when p_L is set to 0. However, in messaging, DSAN is an order of magnitude more costly. This is where our paper stresses that we are looking at the relative improvement of efficiency when you adjust how often you take a lateral move as opposed to comparing algorithms against one another. The results with DSA-B and DSAN show that the same effect can be achieved – reducing the messaging cost is possible without affecting solution quality in a significant way. The relative importance of messaging cost vs. solution quality for the system will influence how much of a tradeoff is permissible.

We find that DBA is so heavy-laden with messaging costs (hundreds of thousands compared to a couple thousand) compared to DSA that it is orders of magnitude less efficient. Opposite to the DSA variants, DBA, MGM, and DisPeL use the same number of messages, indicating that optimizing escape strategies to get better solutions is worth the effort. This is highlighted by the efficiency metric, showing that MGM has much lower efficiency because it did not allow any escape from local minima whereas in both DBA and DisPeL, the efficiencies were competitive. If we had chosen to continue letting the simulations run, it is possible that DisPeL would have come up with superior solutions to DBA as has been shown in the past, however the premise of this paper is to show the importance of lateral movement escape strategies in different families of algorithms. From our results, we show that improving the escape strategy in algorithms related to DBA where messaging is fixed will improve the algorithm's efficiency, whereas in DSA types of algorithms, the escape strategy is much more costly.

VI. CROSS VALIDATION

Additionally, we also ran these same tests in a sensor tracking environment to see if our conclusions were specific to graph coloring or would be reproduced across domains. To cross validate our findings in a more realistic domain, we decided to test our variants using a polynomial solvable resource allocation problem that assigns sensor to targets in a sensor network. Figure 7 shows a similar trend in the sensor tracking domain, indicating that different domains still battle the payoff of lateral movement.

A. Definition

The tracking domain is an implementation of the complete-compatibility version of the SensorDCSP formulation presented in [**?**], [**?**], [**?**]. In this domain, there are a fixed number of sensors and targets randomly placed within an environment. Because of range restrictions, only sensors that are within some distance $dist$ can see a target. Because of the need to triangulate the targets, three sensors are needed

Algorithm	RES	TMR	EFF	REM
DSA-B (a)	159.3	1313	0.12	2.7
DSA-B (b)	159.4	540	0.30	2.6
DSA-B (c)	158.8	138	1.15	3.2
DSA-B1 (d)	159.62	264	0.60	2.4
DSA-B1E (e)	159.1	137	1.16	2.9
DSA-B1E (f)	158.8	133	1.19	3.2
DSAN	156.7	10694	0.014	5.3
DSANE (g)	159.94	3210	0.049	2.1
DSANE (h)	159.4	1890	0.084	2.6
DBA	156.2	25616	0.00609	5.8
MGM-1	155.78	25616	0.00608	6.2
DisPeL	152.1	25616	0.00593	9.9

TABLE IV
RESULTS/CONFIGURATIONS: CONFLICTS RESOLVED (RES), TOTAL MESSAGES RECEIVED (TMR), EFFICIENCY (EFF), AND CONFLICTS REMAINING (REM) FOR 224 SENSORS AND 34 TARGETS IN THE SENSOR TRACKING DOMAIN.

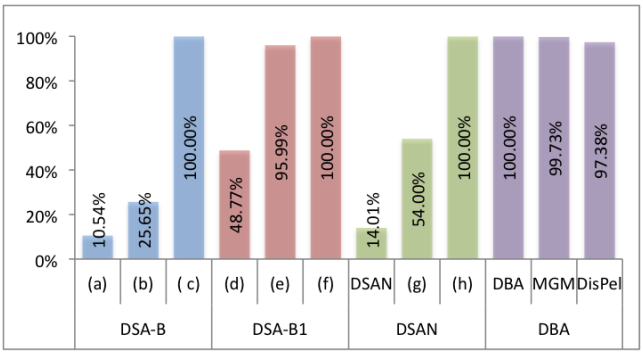

Fig. 5. Groupwise Comparison of Efficiency per algorithm for the tracking domain.

to accurately localize their positions. A cost occurs when a sensor is assigned to two targets at the same time. The goal is to find an assignment of sensors to targets such that each target has three sensors tracking while minimizing any overlap.

Following directly from the definition for a DCOP, a SensorDCOP problem consists of the following:

- a set of n targets $T = \{T_1, \ldots, T_n\}$.
- a set of g agents $A = \{a_1, \ldots, a_g\}$.
- a set of possible sensors that can "see" each of the targets $D = \{D_1, \ldots, D_n\}$
- a set of cost functions $F = \{f_1, \ldots, f_m\}$ where each $f_i(v_i, v_j)$ is function that returns $|v_i \cap v_j|$.

The problem is to find an assignment $S^* = \{v_1, \ldots, v_n\}$ such that $F(S) = \sum_{i=1}^{m} f_i(S)$ is minimized and each v_i is a set of $\binom{|D_i|}{c}$ sensors from D_i where $c = min(|D_i|, 3)$. This indicates that each target requires three sensors, if enough are available, or all of the sensors, if there are less than three. Since, in this implementation, each of the sensors is compatible with one another, the overall complexity of the problem is polynomial as was shown in [**?**], using a reduction to feasible flow in a bipartite graph.

B. Setup and Results

To test our DSA variations in the tracking domain, we ran a test series which used a 200 meter $\times$ 200 meter environment with 224 sensors placed in an ordered grid-based pattern. In these tests, each sensor is able to "see" a target that is within 20 meters of its position. We chose to place the sensors in an ordered fashion to reduce the variance obtained within the results. We ran a test series which varied the sensor-to-target ratio from 10:1 to 3.5:1 in increments of 1 which is across the spectrum from sparsely populated, underconstrained to densely populated, overconstrained instances. We then conducted 150 trial runs with a random target placement for each of these configurations to get a good statistical sampling. The random seeds used to place the targets were saved, so all the variants were tested using identical problem instances. We ran each trial for 300 cycles.

Table IV shows the results for the tracking domain. In comparison to the graph coloring domain, the efficiency trends are quite similar (see Figure 5), although the differences between each algorithm in a group is more pronounced.

VII. DEEPER ANALYSIS

We wanted to figure out why such dramatic improvement was shown in the DSA variants, so we conducted another set of runs using DSA-BE to include counts of lateral moves taken in comparison to the number of hill-climbing moves. Figure 8 shows the average number of times an agent with four neighbors attempts to make moves that result in improve values of zero (lateral moves) and improve values greater than zero (hill climbing moves). As you can see, lateral moves dominate by a sufficient margin, indicating that adjusting p_L should affect the number of messages more than changing p_H.

Once we discovered this phenomena, we tested all combinations of $p_L = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and $p_H = \{0.3, 0.5\}$ in both the graph coloring and sensor tracking domains to see if there was a direct correlation to the algorithm's efficiency. We found that as the probability for making an escape move increases (p_L), the efficiency decreases, as shown in Figure 6. To be sure this wasn't affected by changing the value of p_H, we also used another p_H value and saw the same trend.

VIII. CONCLUSION

For more efficient algorithms, like DSA variants, limiting or eliminating the escape strategy still allows the algorithm to arrive at an almost identical solution while significantly reducing cost. After examining the results from DSA, DSAN, and DSA-B1, we see that the effect of lateral escape moves is the same: a lot of additional cost for little gain. The answer is different for DBA. Because the algorithm uses a constant number of messages, an escape strategy that eventually yields a better solution actually improves the algorithm.

So, do you get what you pay for when exploring the search space using DSA-like algorithms? The answer is no unless messaging cost is inconsequential or solution quality is paramount. The messages passed are much more effective when you do not allow local minima to be escaped. However, if an escape strategy is desired, simply reducing the probability

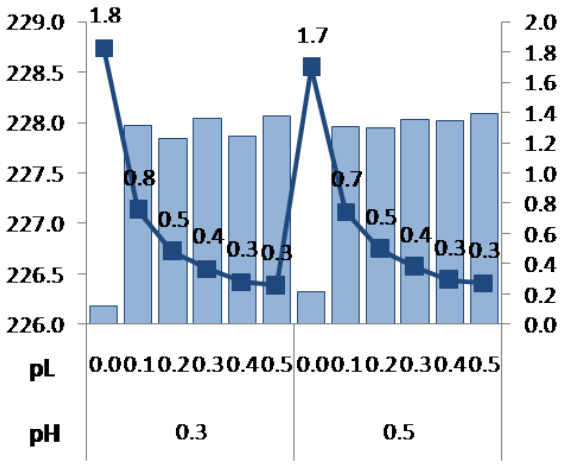

Fig. 6. Overall conflicts resolved (bar) and efficiency (line) for DSA graph coloring agents for 300 nodes at 2.3n density (within the phase transition).

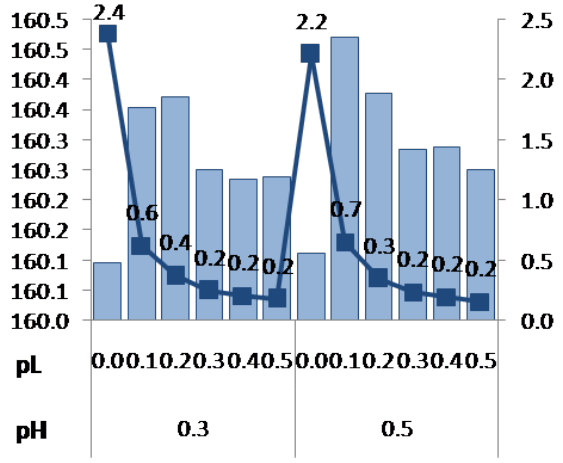

Fig. 7. Overall conflicts resolved (bar) and efficiency (line) for DSA sensor tracking agents for tracking 34 targets with 224 sensors (within the phase transition).

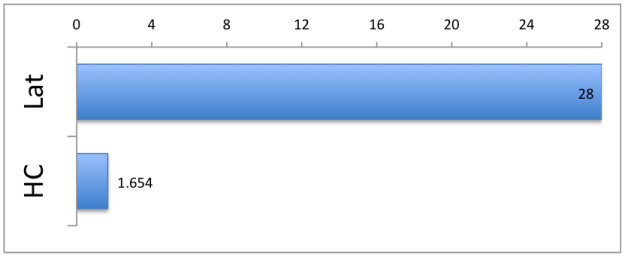

Fig. 8. Average number of Lateral Moves (Lat) vs. Hill-Climbing Moves (HC) for graph-coloring agents with four neighbors.

of making an escape move can significantly reduce the number of messages while achieving a better solution than if no escape strategy was employed.

ACKNOWLEDGMENT

We would like to thank Mike Michalak for his invaluable help coding the algorithms tested in this paper.

The authors gratefully acknowledge support of the Defense Advanced Research Projects Agency under DARPA grants HR0011-07-C-0060. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA.

REFERENCES

[1] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method for distributed constraint optimization," in *2nd Int'l Joint Conf. on Autonomous Agent and Multiagent Systems*, Melbourne, Australia, July 2003.

[2] R. Mailler and V. Lesser, "Solving distributed constraint optimization problems using cooperative mediation," in *AAMAS*, 2004, pp. 438–445.

[3] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," in *IJCAI*, 2007.

[4] M. Yokoo and K. Hirayama, "Distributed breakout algorithm for solving distributed constraint satisfaction problems." in *Int'l Conf. on Multiagent Systems*, 1996, pp. 401–408.

[5] P. Morris, "The breakout method for escaping local minima," in *Nat'l Conf on Artificial Intelligence*, 1993, pp. 40–45.

[6] R. T. Maheswaran, J. P. Pearce, and M. Tambe, "Distributed algorithms for dcop: A graphical game-based approach," in *17th Int'l Conf. on Parallel and Dist. Computing Systems*, 2004.

[7] M. Yokoo and E. H. Durfee, "Distributed constraint optimization as a formal model of partially adversarial cooperation," University of Michigan, Ann Arbor, MI 48109, Tech. Rep. CSE-TR-101-91, 1991.

[8] F. Bacchus and P. van Beek, "On the conversion between non-binary constraint satisfaction problems," *15th Nat'l/10th Conf. on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pp. 311–318, 1998.

[9] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Ch 13: A comparative study of distributed constraint algorithms," *Distributed Sensor Networks: A Multiagent Perspective*, 2003.

[10] W. Zhang, G. Wang, and L. Wittenburg, "Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance," *AAAI Wksp on Probabilistic Approaches in Search*, 2002.

[11] M. Arshad and M. Silaghi, "Distributed Simulated Annealing and comparison to DSAtrade-offs betweenanytimeand marginally improved quality, local effort and communication," *IJCAI Wksp on Dist. Constraint Reasoning*, 2003.

[12] J. P. Pearce and M. Tambe, "Quality guarantees on k-optimal solutions for distributed constraint optimization problems," in *Int'l Joint Conf. on Artificial Intelligence*, 2007.

[13] H. Katagishi and J. P. Pearce, "Kopt : Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility," in *Wksp. on Dist. Constraint Reasoning*, 2007.

[14] M. Basharu, I. Arana, and H. Ahriz, "Escaping local optima: Constraint weights vs. value penalties," *AI*, 2007.

[15] R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman, "Distributed constraint satisfaction in a wireless sensor tracking system," in *Wksp on Dist Constraint Reasoning*, 2001.

[16] B. Krishnamachari, R. Bejar, and S. Wicker, "Distributed problem solving and the boundaries of self-configuration in multi-hop wireless networks," in *P35th Hawaii Int'l Conf. on System Sciences*, 2002.

[17] C. Fernandez, R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman, "Ch. 12: Communication and computation in distributed csp algorithms," *Distributed Sensor Networks: A Multiagent Perspective*, 2003.

[18] B. Krishnamachari, "Phase transitions, structure, and complexity in wireless networks," Ph.D. dissertation, Cornell University, 2002.