Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance*

Weixiong Zhang, Guandong Wang and Lars Wittenburg
Department of Computer Science
Washington University
St. Louis, MO 63130
email: zhang@cs.wustl.edu

Abstract

Many distributed problems can be captured as distributed constraint satisfaction problems (CSPs) and constraint optimization problems (COPs). In this research, we study an existing distributed search method, called distributed stochastic algorithm (DSA), and its variations for solving distributed CSPs and COPs. We analyze the relationship between the degree of parallel executions of distributed processes and DSAs' performance, including solution quality and communication cost. Our experimental results show that DSAs' performance exhibits phase-transition patterns. When the degree of parallel executions increases beyond some critical level, DSAs' performance degrades abruptly and dramatically, changing from near optimal solutions to solutions even worse than random solutions. Our experimental results also show that DSAs are generally more effective and efficient than distributed breakout algorithm on many network structures, particularly on over-constrained structures, finding better solutions and having lower communication cost.

Introduction

In recent years, various micro-electro-mechanical systems (MEMS) devices, such as sensors and actuators with some information processing capabilities embedded within, have been developed and deployed in many real-world applications [9; 10]. Multiagent system (MAS) technology can play critical roles in large-scale networked, embedded systems using such smart devices, by providing frameworks for building and analyzing such systems. Due to the real-time nature of many applications and limited computational resources on the devices, e.g., slow CPUs and small memories, the key to large-scale, real-time MEMS is the mechanism that the agents use to make viable distributed decisions in restricted time with limited computational resources.

There are many affecting factors restricting what agents atop of MEMS devices can do. Such factors include the communication reliability and delay, the dynamics of underlying applications, the limited computational resources of individual devices and the requirement of real-time performance. These restrictions imply that complex methods, such as negotiation, that require a substantial amount of computation and communication, are not the right choices for distributed systems with resource-limited devices. The current advances in the MEMS technologies and their real-world applications [9; 10] have manifested many such distributed, real-time situations where complex problem-solving methods are simply infeasible or inappropriate. In a resource-limited distributed system that operates in real-time environments, there is a need for simple methods that have low overheads on computation and communication. It is also desirable that such methods be able to provide good anytime performance.

Examples of such simple, low-overhead methods include the fixed point method [3], distributed breakout [12; 13; 14], and distributed stochastic search [2; 4]. These approaches are sometimes the only feasible methods to address problems in distributed environments, and they may also increase the effectiveness and efficiency of overall problem-solving processes. Moreover, they provide agents with autonomy and degree of parallel executions. Finally, and most importantly, they are simple and require little computation and communication resources. In short, they are the choices of algorithms for multiagent systems controlling small MEMS devices with limited information processing capabilities.

In this research, we study distributed stochastic algorithm (DSA) [4] and its variations for solving distributed constraint satisfaction problems (CSPs) and distributed constraint optimization problems (COPs). Using DSAs, agents may have a high degree of autonomy, making decisions probabilistically, mainly based on local information. The main difference among DSA and its variants is the degree of parallel executions. We experimentally investigate the relationship between the degree of parallelism and the performance of the algorithms.

Motivating Application and Model

In a typical application in the avionics domain in which we are interested, a large number of sensors and actuators are mounted on the surface of an object, such as an aircraft's wing. Such objects may be damaged due to excessive exterior disturbances under certain conditions. The sensors

*This research was funded in part by NSF Grants IIS-0196057 and IET-0111386, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. Thanks to Stephen Fitzpatrick and Zhao Xing for many helpful discussions.

and actuators are arranged in grid structures in which the neighboring sensors and actuators are fixed. To detect possible damages, selected actuators generate signals, in a form of vibrations, to their neighborhoods. The sensors within a neighborhood then detect possible damages at their locations by measuring the frequencies and strengths of the signals. Two restrictions on the system make the problem of damage detection difficult. First, the sensors and actuators operate on limited energy sources, i.e., batteries. Therefore, the set of signaling actuators, which are called *ping nodes*, must be as small as possible, as long as it maintains coverage of the overall area. Second, two signals whose strengths are above a certain threshold at a sensor location will interfere with each other. This constraint, therefore, requires that the signaling actions of two overlapping ping nodes be synchronized so that no interfering signals will be generated at a sensor location at any time.

We are developing a large-scale agent system for this damage detection application. In our system, we embed an agent in each sensor and actuator to control its sequence of actions. These embedded, distributed agents then collaboratively detect possible damage to the area the sensors cover using a small amount of energy and with a low real-time response time. We need to make our system scalable to accommodate the restricted resources on the underlying hardware and to meet the real-time requirement. Therefore, we make the agents as autonomous as possible by distributing all the decision-making functionalities to individual units. Simple, low-overhead methods are then adopted to reduce communication costs and to speed up decision making processes.

This damage detection problem can be captured by a constraint model. Scheduling the signaling activities of the ping nodes can be formulated as a distributed graph coloring problem. A color here corresponds to a time slot in which a ping node sends out signals. The number of colors is therefore the length in time units of a schedule. The problem is to find a shortest schedule such that the pinging signals do not interfere with one another in order to increase damage detection response time and reduce the amount of wasted energy. The problem is equivalent to finding the chromatic number of a given constraint graph, which corresponds to the minimal worst-case response time and a coloring of the graph within the overall system response time.

In short, this damage detection problem is a distributed constraint satisfaction/optimization problem with variables and constraints distributed among agents. Collectively the agents find a solution to minimize an objective function, which is the number of violated constraints in our study.

Distributed Stochastic Search

Distributed stochastic algorithm (DSA) is uniform [11], in that all processes are equal and have no identities to distinguish one another. It is also synchronous in principle [11], in that all processes proceed in synchronized steps and in each step it sends and receives (zero or more) messages and then performs local computations, i.e., changing local state. Note that synchronization in DSA is not crucial since it can be achieved by a synchronization mechanism [11].

Algorithm 1 Sketch of DSA, executed by all agents.

```
Randomly choose a value
while (no termination condition is met) do
  if (a new value is assigned) then
    send the new value to neighbors
  end if
  collect neighbors' new values, if any
  select and assign the next value (See Table 1)
end while
```

Algo.	$\Delta > 0$	C, $\Delta = 0$	no C, $\Delta = 0$
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 1: Next value selection in DSAs. Here C stands for conflict, Δ is the best possible conflict reduction between two steps, v the value giving Δ, and p a probability to change the current value, which represents the degree of parallel executions, and "-" means no value change. Notice that when $\Delta > 0$ there must be a conflict.

The idea of DSA and its variations is simple [2; 4]. After an initial step in which the agents pick random values for their variables, they go through a sequence of steps until a termination condition is met. In each step, an agent sends its current state information, i.e., its variable value in our case, to its neighboring agents if it changed its value in the previous step, and receives the state information from the neighbors. It then decides, often stochastically, to keep its current value or change to a new one. The objective for value changing is to possibly reduce violated constraints. A sketch of DSA is in Algorithm 1.

The most critical step of DSA is for an agent to decide the next value, based on its current state and its believed states of the neighboring agents. If the agent cannot find a new value to improve its current state, it will not change its current value. If there exists such a value that improves or maintains state quality, the agent may or may not change to the new value based on a stochastic scheme.

Table 1 lists five possible strategies for value change, leading to five variations of the DSA algorithm. In DSA-A, an agent will change its value only when the state quality can be improved. DSA-B is the same as DSA-A except that an agent may also change its value if there is a violated constraint and changing its value will not degrade state quality. DSA-B is expected to have a better performance than DSA-A since by reacting stochastically when the current state cannot be improved directly ($\Delta = 0$ and there exists a conflict), the violated constraint may be satisfied in the next step by the value change at one of the agents involved in the constraint. Thus, DSA-B will change value more often and has a higher degree of parallel actions than DSA-A.

Furthermore, DSA-C is more aggressive than DSA-B, changing value even if the state is at a local minima where

there exist no conflict but another value leading to a state of the same quality as the current one. An agent in DSA-C may move to such an equal-quality value in the next step. It is hoped that by moving to another value, an agent gives up its current value that may block any of its neighbors to move to a better state. Therefore, the overall quality of the algorithm may improve by introducing this equal-quality action at a single node. The actual effects of this move remain to be examined, which is one of the objectives of this research.

Parallel to DSA-B and DSA-C, we have two more aggressive variations. DSA-D (DSA-E) extends DSA-B (DSA-C) by allowing an agent to move, deterministically, to a new value as long as it can improve the current state ($\Delta > 0$). These variations make an agent more greedily self centered in that whenever there is a good move, it will take it.

Notice that the level of activities at an agent increase from DSA-A, to DSA-B and to DSA-C, and from DSA-D to DSA-E. The level of activities also reflects the degree of parallel executions among neighboring processes. When the level of local activities is high, so is the degree of parallel executions.

To change the degree of parallel executions, an agent may switch to a different DSA algorithm, or change the probability p that controls the likelihood of updating its value if the agent attempts to do so. This probability controls the level of activities at individual agents and the degree of parallel executions among neighboring processes. One major objective of this research is to investigate the effects of this control parameter on the performance of DSA algorithms.

The termination conditions and methods to detect them are complex issues of their own. We will adopt a termination detection algorithm [11] in a later stage. In our current implementation, we terminate DSAs after a fixed number of steps. This simple determination method serves the basic needs of the current research, i.e., experimentally investigating the behavior and performance of these algorithms, the main topic of this paper.

Experiment Setup

In our experiments, we used different networks, including grids, which appear in our motivating application, and graphs and trees. We considered graph coloring problems, by varying the connectivity of the structures and the number of colors used, we are able to generate underconstrained, critically constrained and overconstrained problem instances. In the following discussions, we will focus on grid and graph structures.

We generate grids of various sizes, including 20×20, 40×40 and 60×60 grids, and use different number of colors, ranging from two to eight. In order to study how DSAs will scale up to large problems, we simulate infinitely large grids. We remove the grid boundaries by connecting the nodes on the top to those on the bottom as well as the nodes on the left to those on the right of the grids. We also change the degree of constrainedness by changing the number of neighbors that a node may have. For example, on a degree $k = 4$ grid, each node has four neighbors, one each to the top, the bottom, the left and the right. Similarly, on a degree $k = 8$ grid, each node has eight neighbors, one each to the top left, top right, bottom left and bottom right in addition to the four neighbors in a $k = 4$ grid.

We generate graphs with 400 and 800 nodes and average node connectivity equal to $k = 4$ and $k = 8$. A graph is generated by adding edges to randomly selected pairs of nodes. These two types of graphs are used to make a correspondence to the grid structures of $k = 4$ and $k = 8$ mentioned before. We also generated random trees with depth four and average branching factors $k = 4$ and $k = 8$.

The distributed algorithms were simulated on one machine using a discrete event simulation method [12]. In this method, an agent maintains a step counter, equivalent to a simulated clock. The counter is increased by one after the agent has executed one step of computation, in which it sends its state information, if necessary, receives neighbors' messages, and carries out local computation. The overall solution quality is measured, at a particular time point, by the total number of constraints violated, and the communication cost is measured by the total number of messages sent.

Phase Transitions

DSAs are stochastic, in that they may behave differently even if all conditions are equal. We are interested in their typical or statistical behavior at an equilibrium when the behavior of the algorithms does not seem to change dramatically from one step to the next. We are specifically interested in the relationship between the degree of parallel executions, controlled by the probability p (cf. Table 1), and the performance of the algorithms, including their solution quality and communication costs.

It turns out that the performance of DSAs may experience phase transitions on some constraint structures when the degree of parallelism increases. Phase transitions refer to a phenomenon of a system in which some global properties change rapidly and dramatically when a control or order parameter goes across a critical value [1; 6]. A simple example of a phase transition is water changing from liquid to ice when the temperature drops below the freezing point. For the problem of interest here, the system property is DSAs performance (solution quality and communication cost) and the order parameter is the probability p that controls the degree of parallel executions of the agents.

Phase transitions on solution quality

We experimentally investigate DSAs' phase-transition behavior on grids, random graphs and trees. Starting from random initial colorings, we let the algorithms run for a large number of steps, to the point where they seem to reach an equilibrium, i.e., the overall coloring quality does not change significantly from one step to the next. We then measure the solution quality, in terms of the number of constraints violated. In our experiments, we measure the performance at 1,000 steps; longer executions, such as 5,000 and 10,000 steps, exhibit almost the same results.

We varied the degree of parallel executions, the probability p in Table 1, and examined the quality of the colorings that DSAs can provide. The solution quality indeed exhibits phase-transition behavior on grid and graph structures as the degree of parallelism increases.

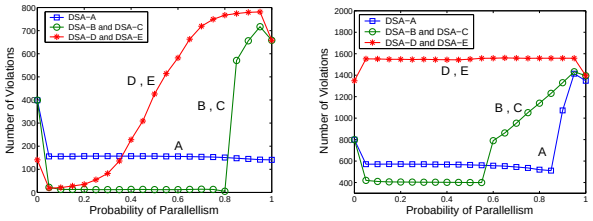

Figure 1: Solution quality phase transitions on 2-coloring grids; $k = 4$ (left) and $k = 8$ (right).

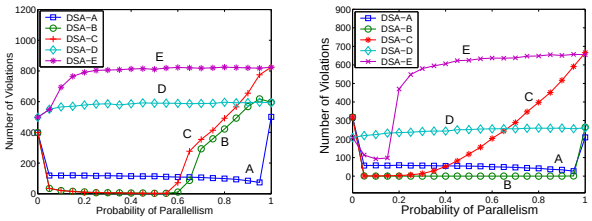

Figure 2: Solution quality phase transitions on grids; $k = 8$ using 4 colors (left) and 5 colors (right).

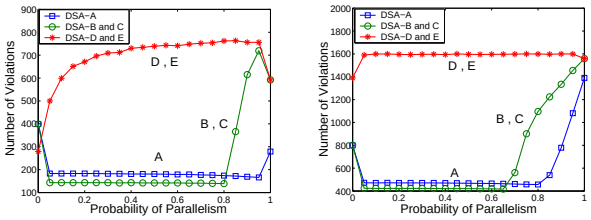

Figure 3: Solution quality phase transitions on 2-coloring graphs; $k = 4$ (left) and $k = 8$ (right).

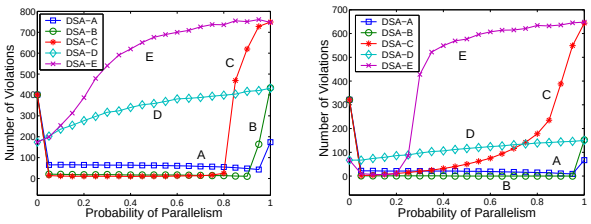

Figure 4: Solution quality phase transitions on graphs; $k = 8$ using 4 colors (left) and 5 colors (right).

Grids We generated problem instances of grids with various sizes as described in Section . Figure 1 shows the total numbers of constraint violations after 1,00 steps of the algorithms using two colors on 20×20 grids with $k = 4$ (Figure 1(left)) and $k = 8$ (Figure 1(right)). Each data point of the figure is averaged over 1,000 random initial colorings. Note that the results from larger grids, such as 40×40 grids, follow almost identical patterns as in Figure 1.

The figures show that DSAs' phase-transition behavior is controlled by the degree of parallelism, except DSA-A on grids with $k = 4$. The transitions are typically very sharp. For example, as Figure 1(left) shows, the solution quality of DSA-B and DSA-C decreases abruptly and dramatically when the probability p increases above 0.8. More importantly and surprisingly, after the transition, the solution quality is even worse than a random coloring. The average solution quality of random colorings corresponds to the point $p = 0$ on the DSA-B and DSA-C curves in the figure. This indicates that the degree of parallel executions should be controlled under a certain level in order for the algorithms to have a good performance. Furthermore, the transitions start earlier for DSA-D and DSA-E. Although DSA-A, the most conservative algorithm, does not show phase transitions on grids of $k = 4$, its average solution quality is much worse than that of DSA-B, because it may be easily trapped in local minima.

The degree of parallelism and the constrainedness of the underlying network structures also interplay. Grids with $k = 4$ are 2-colorable while grids with $k = 8$ are not, and are thus overconstrained. The results shown in Figure 1 indicate that the phase transitions appear sooner on overconstrained problems than on underconstrained problems. Even the most conservative DSA-A also experiences a phase transition when $k = 8$. The most aggressive ones, DSA-D and DSA-E, always performs worse than a random coloring on this overconstrained grid.

A coloring problem becomes easier if more colors are used, since it is less constrained to find a satisficing color in the next step. However, the phase-transition behavior persists even when the number of colors increases. Figure 2 shows the results on grids with $k = 8$ using 4 and 5 colors. Notice that the curves in the 4-color figure and the curves for 3 colors (not shown here) follow similar patterns as in the case for 2 colors in Figure 1(right).

Graphs The phase transitions of DSAs persist on graphs as well, and follow similar patterns as in the grid cases. We conducted experiments on random graphs, with problem instances generated as described in Section . Figure 3 shows the results on graphs with $k = 4$ and $k = 8$ using 2 colors, and Figure 4 the results on graphs with $k = 8$ using 4 and 5 colors. Each data point is an average of 1,000 random instances. The solution quality is also measured after 1,000 steps of executions. As all the figures show, the phase transitions on random graphs have similar patterns as those on grids. Therefore, the discussions on the grids apply in principle to random graphs. We need to mention that on graphs, the most aggressive algorithms, DSA-D and DSA-E, do not perform very well under all degrees of parallel executions. This, combined with the results on grids, leads to the conclusion that DSA-D and DSA-E should not be used.

Trees There is no phase transition observed on random trees in our tests. All DSAs perform poorly on 2-coloring, in comparison with their performance on grids and graphs. This seems to be counterintuitive since trees have the simplest structures among all these network structures. One explanation is that DSAs may be easily trapped into local minima. Since trees are always 2-colorable, we are able to easily

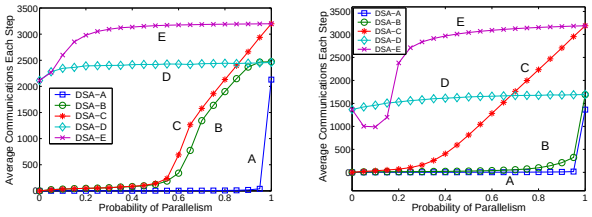

Figure 5: Communication phase transitions on grids with $k = 8$ using 4 colors (left) and 5 colors (right).

create local minima in which none of DSAs can escape.

Phase transitions on communication

We have so far focused on DSAs' solution quality without paying any attention to their communication costs. Communication in a sensor network has an inherited delay and could be unreliable in many situations. Therefore, communication cost of a distributed algorithm is an integral part of its overall performance. It is desirable to keep communication cost as low as possible.

In fact, the communication cost of a DSA algorithm goes hand-in-hand with its solution quality. Recall that an agent will send a message to its neighbors after it changed its value (cf. Algorith 1 and Table 1). In DSA-A, DSA-B and DSA-D, an agent may change its value if there is a conflict, and will not do so if it is currently at a state of a local minimum, while in DSA-C and DSA-E, an agent may probabilistically change its value at a local minimum state. Therefore, in general the communication cost at a node will go down if the agent moves to a better state, and go up otherwise. As a result, the overall communication cost will also follow similar trends. If the solution quality of DSA improves over time, so does its communication cost. Therefore, the communication cost is also controlled by the degree of parallel executions of the agents. The higher the parallel probability p is, the higher the communication cost will be.

We verified this prediction by experiments on grids and graphs, using the same problem instances as used for analyzing solution quality. Figure 5 shows the communication cost on grids with $k = 8$ using 4 colors (left) and 5 colors (right) after 1,000 steps. Comparing these these figures with those in Figure 2, it is obvious that solution quality and communication cost follow identical patterns. Furthermore, the communication cost on graphs (not shown here) follows similar patterns as those on grids.

Since DSAs' communication cost follows their solution quality, in the rest of the paper we will simply consider solution quality.

Anytime Performance

Although the phase-transition results are the determinants in choosing a DSA algorithm that can reach stable states, they do not, however, reveal how DSAs will perform during the processes of reaching stable states. We consider anytime performance of DSAs in this section.

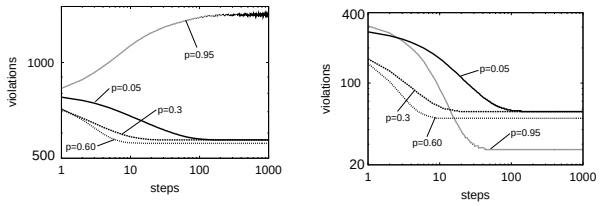

Figure 6: DSA-A on grids $k = 8$; 2 colors (left) and 5 colors (right).

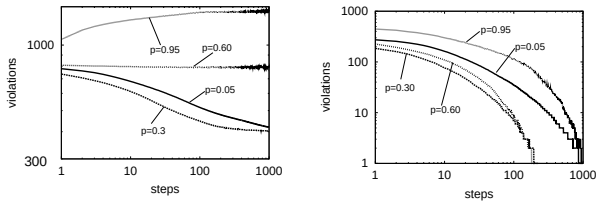

Figure 7: DSA-B on grids $k = 8$; 2 colors (left) and 5 colors (right).

Anytime features are important for a system in dynamic environments in which it may be disastrous to wait for the system to reach stable states. This is particularly true for our damage detection sensor networks since it is desirable to respond to damages as soon as possible. Therefore, we are interested in anytime distributed algorithms, i.e., algorithms that can be stopped at anytime during their executions and are able to provide feasible and high-quality solutions at that point. Fortunately, DSAs can be used for this purpose.

Grids We used the same set of 1,000 problem instances and experimental conditions as in the previous section. For each DSA algorithm, we chose four degrees of parallelism, $p = 0.05$, $p = 0.3$, $p = 0.6$, and $p = 0.95$. The results on grids with $k = 8$ using 2 colors and 5 colors are included in Figures 6 to 9. We plotted the results using logarithmic scales on both the number of steps (the horizontal axes) and the number of violations (the vertical axes) in order to closely examine the anytime progresses of the algorithms.

Recall that DSA-A is the most conservative of the DSA family. As depicted in Figure 6, in general, a higher degree of parallelism should be chosen as long as doing so will not put the algorithm into a degraded region. As shown in Figure 6 for instance, $p = 0.6$ is preferred over $p = 0.3$ and $p = 0.05$. Overall, a middle range p, e.g., $p = 0.6$, seems to be a good parameter to use.

Now consider the results of DSA-B, which is more active than DSA-A. The results are in Figure 7. The phase transition point appears around $p = 0.55$ on $k = 8$ grids using 2 colors (cf. Figure 1(right)). The anytime performance when p is beyond the phase transition point, $p = 0.6$ and $p = 0.95$, is also not competitive. In both cases in Figure 7, $p = 0.3$ gives the best anytime performance, although they all reach optimal solutions after a long run. Notice that on the less constrained grids ($k = 8$ and 5 colors), DSA-B is able to reach the optimal states (Figure 7).

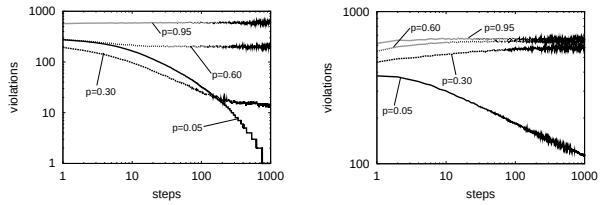

Figure 8: DSA-C (left) and DSA-E (right) on grids $k = 8$ and 5 colors.

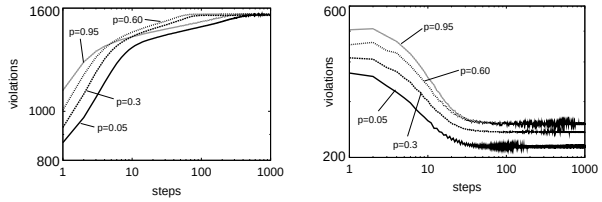

Figure 9: DSA-D on grids $k = 8$; 2 colors (left) and 5 colors (right).

As discussed before, DSA-C behaves the same as DSA-B when using 2 colors on the grids. When using 5 colors, the performance of DSA-C with $p = 0.05$ is compatible with that with $p = 0.3$ before the first 100 steps (Figure 8(left)). Moreover, DSA-C with $p = 0.05$ reaches the optimal solution, indicating a smaller degree of parallelism is a better choice.

Now we come to DSA-D and DSA-E. DSA-D and DSA-E behave the same using 2 colors. Figure 9 shows the results of DSA-D and Figure 8(right) the results of DSA-E using 5 colors. All the results in the figures indicate that in these two aggressive algorithms, the lower the degree of parallel executions, the better.

Graphs The anytime performance of DSAs on random graphs in principle follows similar patterns as those on grids. Again, DSA-C and DSA-E are not very competitive on graphs, so we do not include their results here. In Figures 10 and 11, we show the results of DSA-A, DSA-B and DSA-C on random graphs. The anytime performance of DSA on graphs have similar behavior as that on grids.

In summary, a balance between the inherited aggressiveness of a DSA algorithm and its degree of parallelism must be maintained in order to achieve a good anytime performance as well as a good final solution quality. The more

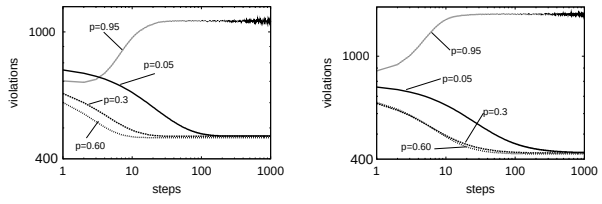

Figure 10: DSA-A (left) and DSA-B (DSA-C) (right) on graphs $k = 8$ and 2 colors.

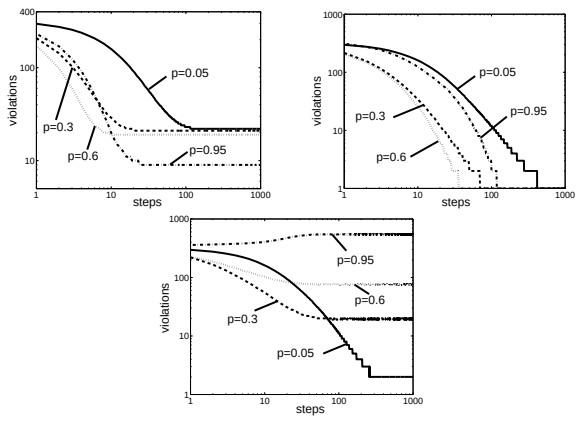

Figure 11: DSA-A (left), DSA-B (middle) and DSA-C (right) on graphs $k = 8$ and 5 colors.

conservative a DSA algorithm is, the higher the degree of parallelism can be. Our results also show that the two most aggressive algorithms, DSA-D and DSA-E are not effective and should not be used in these cases.

Trees On trees, DSAs are not effective. As we briefly explained in Section , they may be easily trapped into local minima. The algorithms usually do not find optimal solutions, even if they exist.

Comparison with DBA

The remaining question is how DSAs compare with other distributed algorithms. The closest competing algorithm is distributed breakout algorithm (DBA) [12; 13; 14] that has almost all the desired features. We compare DSAs with DBA in this section.

Again, we only considered grids and graphs in our experiments, because on trees DSA cannot compete with DBA, which finds optimal solutions in $O(n^2)$ steps on trees with n nodes. We also left out DSA-D and DSA-E since they are not competitive. We chose the best parameters for the other DSAs. We used the same set of 1,000 problem instances used before.

Figure 12 shows the comparison results on grids with $k = 8$ using 2 colors and 5 colors. With 2 colors (Figure 12(left)), DSA-A with $p = 0.6$ has a slightly better performance than DBA in the first 100 steps, but converges to worse states than DBA in a long run. DSA-B (also DSA-C) has worse performance than DSA-A and DBA in about the first 20 steps, but does better afterwards. It reaches much better states than DBA after 1,000 steps. With 5 colors (Figure 12(right)), DBA-A with $p = 0.6$ has a good initial performance but does the worst at the end. DSA-B with $p = 0.3$ has worse anytime performance than DBA, but is able to find optimal solutions after 220 steps. DBA failed to find optimal solution mong 2.5% of the 1,000 trials. DSA-C with $p = 0.05$ is not competitive in this case.

As shown in Figure 13, the results on graphs are similar to that on grids in Figure 12. One exception is that on graphs

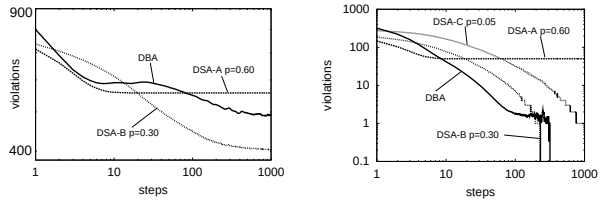

Figure 12: Best DSAs versus DBA on grids $k = 8$; 2 colors (left) and 5 colors (right).

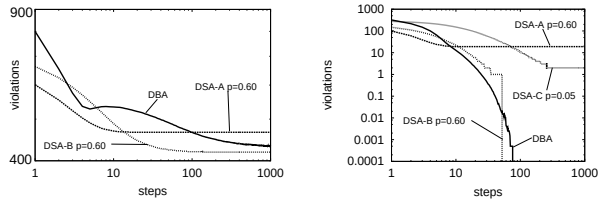

Figure 13: Best DSAs vs. DBA on graphs $k = 8$; 2 colors (left) and 5 colors (right).

with 5 colors, DBA finds optimal solutions at the end.

The results presented in this section show that DSA-B is the best of the DSA family. It is also competitive, especially for finding good or optimal solutions with a long execution.

Related Work and Discussions

The basic idea of distributed stochastic search must have been around for some time. A similar idea was used in distributed belief update [8]. The idea was directly used for distributed graph coloring in [2; 4]. DSA-B considered here is the same as CFP in [4]. However, [2; 4] failed to reveal phase transitions discussed in this paper. The idea was also studied using spin glasses models [7] where phase transitions were characterized. Phase transitions in distributed constraint problem solving was also reported in [5].

This research extends the existing work in many different ways. It proposes two variations to the basic DSA. It systematically studies observation-based, distributed stochastic search for distributed coordination and provides an experimental, qualitative analysis on the relationship among the degree of parallelism, problem constrainedness, solution quality and overall system behavior such as phase transitions. It also demonstrates that phase transitions exist in many different problems and problem structures and they persist when the degree of parallelism changes. Notice that the phase transitions considered in this paper are different from phase transitions of graph coloring problems [1]. Here we studied the phase-transition behavior of distributed search algorithms, which needs not be phase transitions of the coloring problems we considered.

Conclusions

Motivated by real applications of multiagent systems in sensor networks, we studied low-overhead distributed stochastic algorithms (DSAs) for solving distributed constraint satisfaction and optimization problems. We specifically investigated the relationship among the degree of parallel executions, constrainedness of underlying problems, and DSAs' behavior and performance. In addition to showing the phase transitions of solution quality of DSAs on different constraint network structures, our experimental results also lead to two conclusions. First, a very high degree of parallel executions may not be helpful. Very often it may lead to degenerated system performance. An algorithm having a very high degree of parallel executions may even produce results worse than random solutions. Second, a moderate degree of parallel executions may be able to provide high quality global solutions. Indeed, DSA-B outperforms distributed breakout algorithm on many overconstrained problems.

References

[1] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91*, pages 331–337.

[2] M. Fabiunke. Parallel distributed constraint satisfaction. In *Proc. Intern. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA-99)*, pages 1585–1591, 1999.

[3] M. Fenster, S. Kraus, and J. Rosenschein. Coordination without communication: Experimental validation of focal point techniques. In *Proc. ICMAS-95*, 1995.

[4] S. Fitzpatrick and L. Meertens. An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In *Proc. 1st Symp. on Stochastic Algorithms: Foundations and Applications*, pages 49–64, 2001.

[5] K. Hirayama, M. Yokoo, and K. Sycara. The phase transition in distributed constraint satisfaction problems: First results. In *Proc. Intern. Workshop on Distributed Constraint Satisfaction*, 2000.

[6] T. Hogg, B. A. Huberman, and C. Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.

[7] W. G. Macready, A. G. Siapas, and S. A. Kauffman. Criticality and parallelism in combinatorial optimization. *Science*, 271, 1996.

[8] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:245–257, 1987.

[9] H. Reichl. Overview and development trends in the field of MEMS packaging. invited talk given at 14th Intern. Conf. on Micro Electro Mechanical Systems, Jan. 21-25, 2001, Switzerland.

[10] M. Takeda. Applications of MEMS to industrial inspection. invited talk, 14th Intern. Conf. on Micro Electro Mechanical Systems, Jan. 21-25, 2001.

[11] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[12] M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer Verlag, 2001.

[13] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proc. ICMAS-96*.

[14] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proc. AAAI-02, to appear*.