Available online at www.sciencedirect.com

SCIENCE DIRECT®

Artificial Intelligence 161 (2005) 55–87

Artificial Intelligence

www.elsevier.com/locate/artint

Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks

Weixiong Zhang [*], Guandong Wang [1], Zhao Xing [1], Lars Wittenburg [1]

Department of Computer Science and Engineering, Washington University in Saint Louis, Saint Louis, MO 63130, USA

Received 21 April 2003; accepted 1 October 2004

Abstract

This research is motivated by some distributed scheduling problems in distributed sensor networks, in which computational and communication resources are scarce. We first cast these problems as distributed constraint satisfaction problems (DisCSPs) and distributed constraint optimization problems (DisCOPs) and model them as distributed graph coloring. To cope with limited resources and restricted real-time requirement, it is imperative to use distributed algorithms that have low overhead on resource consumption and high-quality anytime performance. To meet these requirements, we study two existing DisCSP algorithms, distributed stochastic search algorithm (DSA) and distributed breakout algorithm (DBA), for solving DisCOPs and the distributed scheduling problems. We experimentally show that DSA has a phase-transition or threshold behavior, in that its solution quality degenerates abruptly and dramatically when the degree of parallel executions of distributed agents increases beyond some critical value. We also consider the completeness and complexity of DBA for distributed graph coloring. We show that DBA is complete on coloring acyclic graphs, coloring an acyclic graph of n nodes in $O(n^2)$ steps. However, on a cyclic graph, DBA may never terminate. To

[*] Corresponding author. Phone: (314)935-8788.
E-mail address: zhang@cse.wustl.edu (W. Zhang).
[1] These authors contributed equally to this research.

0004-3702/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2004.10.004

improve DBA's performance on coloring cyclic graphs, we propose two stochastic variations. Finally, we directly compare DSA and DBA for solving distributed graph coloring and distributed scheduling problems in sensor networks. The results show that DSA is superior to DBA when controlled properly, having better or competitive solution quality and significantly lower communication cost than DBA. Therefore, DSA is the algorithm of choice for our distributed scheduling problems and other distributed problems of similar properties.

Keywords: Distributed constraint satisfaction problem; Distributed constraint optimization problem; Distributed sensor networks; Distributed stochastic search; Distributed breakout algorithm; Breakout algorithm; Phase transitions; Distributed scheduling

1. Introduction and overview

In recent years, various micro-electro-mechanical systems (MEMS) devices, such as sensors and actuators with information processing capabilities embedded within, have been developed and deployed in many real-world applications [12,13]. For example, one application that we are interested is detecting and tracking mobile objects in dynamic and real-time environments, where distributed sensors must cooperatively monitor an area to detect new objects and some of the sensors have to sense and measure at the same time in order to triangulate a target. Another application is acoustic fairing for vibration dampening in the avionics domain using a grid of distributed sensors and actuators, whose actions have to be coordinated properly. In all these applications, there are logical restrictions among the actions of the sensors and actuators and restrictions on available computational and communication resources as well as restrictions on energy consumption.

Multi-agent system technology can play a critical role in developing 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 smart devices (or agents) use to make viable distributed decisions in restricted time with limited computational resources. Therefore, the methods for distributed constraint problem solving are also important tools for such real-time decision making in distributed environments.

Complementary to the current research on sensor networks, which has largely focused on building *ad hoc* mobile communication networks (e.g., [5,23]), this line of research of applying multi-agent system technology and constraint problem-solving techniques has a paramount importance to the overall success of embedded sensor networks by addressing the key issue of resource allocation at the application level.

We have been developing large-scale sensor networks controlled by distributed multi-agent systems for the applications of mobile object detection and tracking problem and the vibration dampening in the avionics domain, which we formulate as distributed scheduling with various constraints in the forms of distributed constraint satisfaction problems (DisCSPs) and distributed constraint optimization problems (DisCOPs). We place an agent on top of a sensor, which runs on a battery and has a slow CPU, a small memory and a wireless communication unit. The sensors/agents need to be programmed in such a way that they can collaboratively carry out a task, which no single sensor/agent can finish on its own. In this research, we are particularly interested in *low-overhead methods* for solving DisCSPs and DisCOPs. By low-overhead methods we mean those in which agents require limited communication and make their own decisions without knowledge of the overall system and the actions of other agents.

In this paper, we study distributed stochastic algorithm (DSA) [2,3,8] and distributed breakout algorithm (DBA) [10,16,18], two existing algorithms for DisCSP, for the purpose of solving our distributed scheduling problems in sensor networks. In other words, we apply and extend these algorithms to solving DisCOPs, which are much more difficult than DisCSPs since an optimization problem is in general harder than its decision counterpart. Our objectives are twofold. First, we aim at understanding how these two algorithms compare with each other on a real-world application such as our scan scheduling problem. Second, we want to identify applicable conditions and parameters of these algorithms for real applications in sensor networks.

Note that DSA and DBA are incomplete; they do not guarantee to find satisfying solutions to DisCSPs and thus may not find optimal solutions to DisCOPs either. Nevertheless, we choose these algorithms for two reasons. First, compared to complete distributed algorithms, such as those of [9,16,17], DSA and DBA do not require any global control mechanism. Agents in these algorithms only need information local to themselves and information of neighboring agents connected through an underlying constraint graph. Therefore, they have low computational and communication cost. These algorithms also do not keep track of many previous states that have been visited, so that memory requirement is low as well. Second, as we will see later in the paper, these algorithms are able to make progressively improving global solutions, which are aggregated from the current states of individual agents. Even though individual agents are not aware of such global states as usually no global state is computed, such states can be used as anytime solutions. Obviously these algorithms do not guarantee the optimality of solutions that they provide, they are sufficient for finding good approximate solutions in real-time environments where the targeting objectives may very often change over time.

In addition to the above important characteristics to serve the needs of some applications in sensor networks, DSA and DBA also have unique features in resolving conflicts among distributed agents. DBA introduces priorities among agents' actions for conflict resolution, while DSA uses randomness to possibly avoid conflicts. Moreover, as shown in this paper, DSA has a threshold behavior, in which solution quality degrades abruptly and dramatically as the randomness for conflict resolutions increases beyond a critical value; and DBA becomes complete and has a low polynomial time complexity for distributed graph coloring problems when the underlying constraint graphs are acyclic. Furthermore, we also show that on other constraint graph structures, including the constraint graphs constructed from our motivating applications, DSA is superior to DBA, generally finding better solutions with less time. Overall, our experimental results show that DSA is a good choice of algorithm for our sensor network applications and other distributed graph coloring problems.

The paper is organized as follows. A detailed account of the scheduling problems in sensor networks is given in Section 2. We formulate the problems as weighted coloring problems in Section 3 and discuss the benchmark problems and simulation methods adopted in this study in Section 4. We then describe DSA and analyze its threshold behavior in Section 5. We consider DBA and its completeness and complexity for graph coloring in Section 6. We also discuss introducing randomness to improve DBA's performance. In Section 7, we compare the performance of DSA and DBA on distributed graph coloring and our distributed scheduling problem of finding the shortest scan schedules for mobile object detection. We discuss related work in Section 8, and finally conclude in Section 9.

Early results of this research have appeared in [19–22].

2. Distributed constraint satisfaction and optimization in sensor networks

To set the stage for our investigation of distributed algorithms,we first describe our application problems in sensor networks, which will be subsequently formulated as distributed constraint problems in the next section.

2.1. Detecting mobile objects

Detecting and tracking mobile objects in large open environments is an important topic that has many real applications in different domains, such as avionics, surveillance and robot navigation. We have been developing such an object detecting and tracking system using MEMS sensors, each of which operates under restricted energy sources, i.e., batteries, and has a small amount of memory and restricted computational power. In a typical application of our system, a collection of small Doppler sensors are scattered in an open area to detect possible foreign objects moving into the region. Each sensor can scan and detect an object within a fixed radius. The overall detecting area of a sensor is divided into three equal sectors, and the sensor can only operate in one sector at any given time. The sensors can communicate with one another through radio communication. The radio channel is not reliable, however, in that a signal may get lost due to, for instance, a collision of signals from multiple sensors, or distorted due to environment noises. Moreover, switching from one scanning sector to another sector and sending and receiving radio signals take time and consume energy. To save energy, a sensor may turn itself off periodically if doing so does not reduce system performance.

The overall system operates to achieve several conflicting goals, to detect as many foreign objects as quickly as possible and meanwhile to preserve energy as much as possible so as to prolong the system's lifetime. The overall problem is thus to find a distributed scan schedule to optimize an objective function that balances the above conflicting goals.

One of our goals here is to develop a scalable sensor network for object detection in that the system response time for detecting a new object does not degenerate when the system size increases. To this end, we place an agent on top of each sensor and make the agent as autonomous as possible. This means that an agent will not rely on information of the whole system, rather on local information regarding its neighboring sensors. These agents also try to avoid unnecessary communication so as to minimize system response time. Therefore, complex coordination and reasoning methods are not suitable and algorithms that have low overhead on computational and communication cost are desirable.

2.2. Acoustic fairing and vibration dampening

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 surface may be distorted due to excessive exterior disturbances. To detect possible excessive disturbances, selected actuators generate signals, in a form of vibrations. The sensors within a neighborhood of an actuator then detect possible damages at their locations by measuring the frequencies and strengths of the vibration signals. To this end, sensors and actuators are arranged in grid structures on the surface to be protected. 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 number of signaling actuators must be as small as possible, as long as they maintain 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 actuators be synchronized so that no interfering signals will be generated at a sensor location at any time.

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 damages to the area the sensors cover using a small amount of energy and with a low real-time response time. We need to make the 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.

3. Constraint models in distributed graph coloring

The problems in sensor networks discussed in the previous section are in essence distributed scheduling problems. Our first step to addressing these problems is to model them as distributed constraint satisfaction and optimization problems. Indeed, the core issues in our applications discussed earlier can be captured as distributed graph coloring problems.

In the mobile object detection problem, a sensor needs to scan its three sectors as often as possible. However, to save energy, two neighboring sensors should try to avoid scanning a common area covered by two overlapping sensor sectors at the same time since one sensor's sensing of an area is sufficient to detect possible objects in the area. The objective is to find a sequence of sensing actions so that each sensor sector can be scanned at least once within a minimal period of time. In other words, the objective is a cyclic scan schedule in which each sector is scanned at least once with the constraint of minimizing the energy usage.

As all agents execute the same procedure, scanning actions on different sensors take approximately the same amount of time. Assume that the sensors are synchronized,[2] an

[2] The algorithms we will consider shortly, DSA and DBA, are synchronous. It has also been observed that under certain conditions asynchronized actions may actually improve the performance of a distributed algorithm [4].

assumption that can be lifted by a synchronization method [14]. Indeed, DBA and its synchronization mechanism have been implemented and tested on a real-time middleware using Object Request Broker framework [15].

We can model the distributed scan scheduling as the following problem of coloring a weighted graph to minimize the weight of violated constraints. Let $G(V, E)$ be an undirected graph with a set of hypernodes V and a set of edges E. Each hypernode $v \in V$, which represents a sensor, consists of k nodes, $v_1, v_2, \ldots, v_k$, which model k sectors of the sensor. In our application, we have $k = 3$. An edge between nodes $u_i \in u$ and $v_j \in v$ is denoted as $(u_i, v_j) \in E$, and u_i and v_j are called neighbors. Two hypernodes are neighbors if any pair of their nodes are neighbors. The weight of an edge (u_i, v_j), denoted as $w(u_i, v_j)$, is the area of a common region covered by sector u_i of sensor u and sector v_j of sensor v. Every hypernode is given a total of T *available colors* to be used, where $|T|$ corresponds to a schedule length when each scanning action is assumed to take a unit time. A sensor may be active and inactive during a scanning process; it is given an activation ratio $\alpha \leqslant 1$ to captures the frequency of how often it should be active. In other words, $1 - \alpha$ is the frequency with which a sensor should turn itself off to save energy. Given the total available colors T, the weighted graph coloring problem is to color the nodes based on the following criteria. First, exactly $\lceil \alpha T \rceil$ colors are used within a hypernode. This means that a hypernode may assign more than one color to a node within the hypernode, but at a different time, if the number of available colors is greater than the number of the nodes in the hypernode, i.e., $\lceil \alpha T \rceil > k$. Second, every node must have at least one color and no two nodes within a hypernode can share a color, meaning that a sensor can only scan one sector at any given time. Third, minimizing conflicts between pairs of nodes in different hypernodes such that the total weight of violations across hypernodes is minimized. When no conflict is allowed or a minimal level of allowed conflicts is specified, the overall problem is to find the minimal number of allowed colors T.

The distortion and damage detection problem discussed in Section 2.2 can also be captured by a constraint model. Scheduling the signaling activities of actuators can be formulated as a distributed graph coloring problem. A color here corresponds to a time slot in which an actuator 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 vibration signals do not interfere with one another 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 the number of violated constraints.

4. Benchmark problems and simulation method

In addition to the above specific graph coloring problems derived from our distributed scheduling problems in sensor network applications, we adopt in this paper three different graph structures in our experimental analysis in order to fully understand the features of the distributed algorithms to be investigated. The first structure is a grid. We generate grids of various sizes, including dimensions of 20×20, 40×40 and 60×60, and use different numbers of colors, ranging from two to eight. In order to study how a distributed algorithm will scale up to large problems, we simulate infinitely large grids by removing the grid boundaries through 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. Specifically, 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. The difficulty of coloring these grids depends on the number of colors used. It is easy to see that all grids with degree $k = 4$ are 2-colorable and all grids with degree $k = 8$ are 3-colorable. In general, coloring a grid of $k = 8$ is more difficult than coloring a grid of $k = 4$ using the same number of colors.

The second graph structure we consider is a random graph. we use 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 degrees of $k = 4$ and $k = 8$ mentioned before. However, as the way these random graphs are generated, their chromatic numbers are generally unknown *a priori*. The chromatic number of a random graph with an average connectivity k is expected to be higher than the chromatic number of a grid with degree k. Therefore, a random graph is typically more difficult to color than a grid if both have the same average connectivity.

The third graph structure we experiment with is a random tree. We generate random trees with depth four and average branching factors $k = 4$ and $k = 8$. Note that trees are 2-colorable.

Using these graph coloring problems and different number of available colors, we are able to generate problem instances of various constrainedness for our experimental analysis.

Following [16], we investigate distributed algorithms on these four sets of graph coloring problems using a discrete event simulation method running on a single machine. 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, if any, and carries out its 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.

5. Distributed stochastic search

In this section, we study the distributed stochastic search algorithm. In fact, this is not a single algorithm, rather a family of distributed algorithms, as we will see shortly.

5.1. The algorithm

Distributed stochastic algorithm (DSA) is uniform [14] in that all processes are equal and have no identities to distinguish one another. It is also synchronous in principle [14] 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 [14]. Under certain conditions asynchronous actions may in fact improve the performance of a DSA algorithm [4].

The idea of DSA and its variations is simple [2,3,8,11]. 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 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 perceived 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 may change its value only when the state quality can be improved. Specifically, when the number of conflicts can be reduced, an agent may change to the value that gives rise to the maximum conflict reduction with probability p; or keep the current value unchanged with probability $1 - p$. DSA-B is the same as DSA-A except that an agent may also change its value with probability p 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 exists no conflict but another value leading to a state of the

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
```

Table 1
Next value selection in DSAs. Here, Δ is the best possible conflict reduction between two steps, v the value giving Δ, and p the probability to change the current value (and thus $1 - p$ the probability to maintain the current value in the next step), and "–" means no value change. Notice that when $\Delta > 0$ there must be a conflict

Algorithm	$\Delta > 0$	Conflict, $\Delta = 0$	No conflict, $\Delta = 0$
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

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 expected 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 swapping at a single node. The actual effects of this move remain to be examined.

Parallel to DSA-B and DSA-C, we have two, more aggressive variations. DSA-D and DSA-E extends DSA-B and DSA-C, respectively, 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 may adopt a distributed termination detection algorithm [14]. Without introducing high computation cost to these low-overhead algorithm, we terminate DSAs after a fixed number of steps in our implementation. This simple termination method serves the basic needs of the current research, i.e., experimentally investigating the behavior and performance of these algorithms, one of the main focuses of this paper.

5.2. Phase transitions

DSAs are stochastic; they may behave differently even if all conditions are equal. We are interested in the relative improvement that these algorithm can make from one step to the next after sufficiently long executions. We are specifically interested in the relationship between the degree of parallel executions, controlled by the probability p (see 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 [1,7]. 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.

5.2.1. Solution quality

We experimentally investigate DSAs' phase-transition behavior on grids, random graphs and trees using the simulation method discussed in Section 4. Starting from random initial colorings, we let the algorithms run for a large number of steps, to the point where 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 1000 steps; longer executions, such as 5000 and 10,000 steps, exhibit almost the same results. We varied the degree of parallel executions, the probability p in Table 1. The solution quality indeed exhibits phase-transition behavior on grid and graph structures as the degree of parallelism increases.

Grids. Fig. 1 shows the total numbers of constraint violations after 1000 steps of the algorithms using two colors on 20×20 grids with $k = 4$ (Fig. 1 (left)) and $k = 8$ (Fig. 1 (right)). Each data point of the figure is averaged over 1000 random initial colorings. Note that the results from larger grids, such as 40×40 grids, follow almost identical patterns as in Fig. 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 Fig. 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

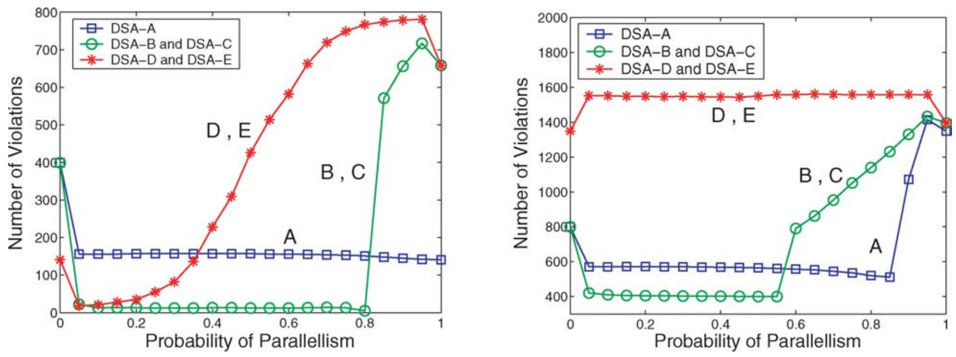


Fig. 1. Solution quality phase transitions on 2-coloring grids; $k = 4$ (left) and $k = 8$ (right).

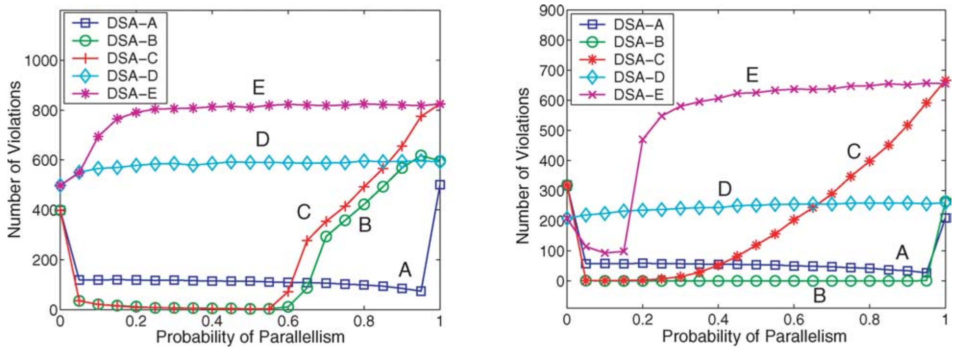


Fig. 2. Solution quality phase transitions on grids; $k = 8$ using 4 colors (left) and 5 colors (right).

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. In contrast, DSA-A, the most conservative algorithm, does not show phase transitions on grids of $k = 4$, and 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 Fig. 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 perform 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 satisfying color in the next step. However, the phase-transition behavior persists even when the number of colors increases. Fig. 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 Fig. 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. We generated graphs with 400 and 800 nodes and averaged node connectivity equal to $k = 4$ and $k = 8$. Fig. 3 shows the results on graphs with $k = 4$ and $k = 8$ using 2 colors, and Fig. 4 the results on graphs with $k = 8$ using 4 and 5 colors. Each data point is an average of 1000 random instances. The solution quality is also measured after 1000 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.

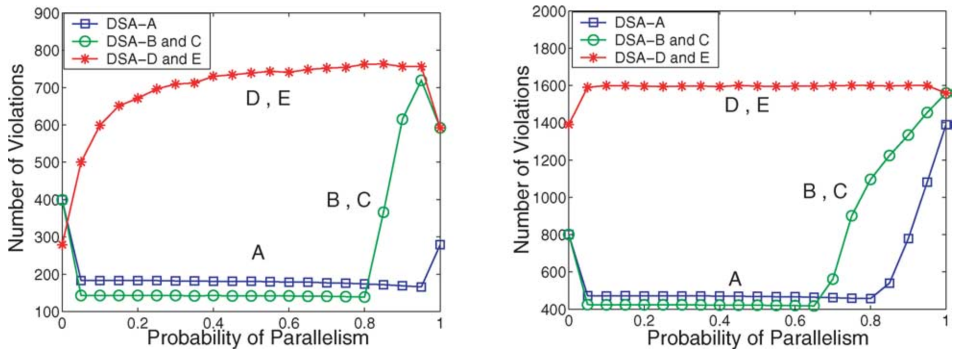


Fig. 3. Solution quality phase transitions on 2-coloring graphs; $k = 4$ (left) and $k = 8$ (right).

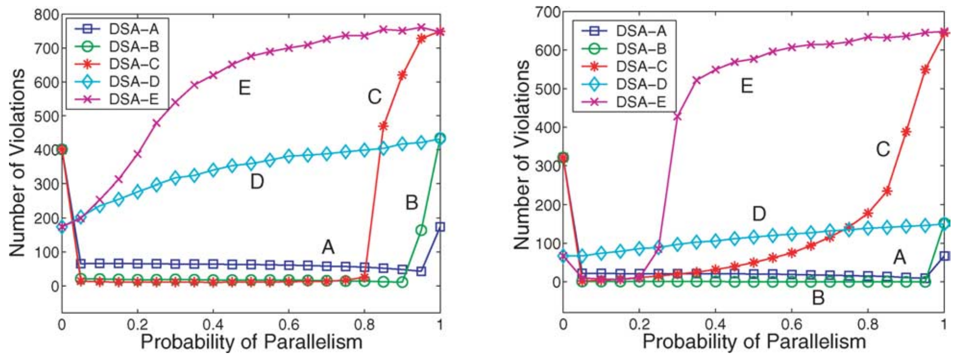


Fig. 4. Solution quality phase transitions on graphs; $k = 8$ using 4 colors (left) and 5 colors (right).

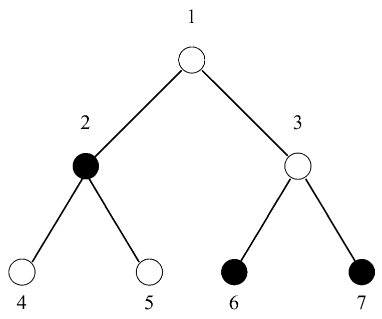


Fig. 5. A local minimum situation for DSA for coloring a tree with 2 colors. See text for detail.

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 counter-intuitive since trees have the simplest structures among all these network structures. A close examination showed that DSAs may be easily trapped into local minima. Fig. 5 illustrates a scenario of local minimum where DSA will not be able to escape. Variables 2 to 7 will not change their values since doing so will increase their conflicts. The only variable that may change value is node 1, which, however, will not improve the solution quality. There are many similar local minima for DSA in coloring trees with two colors.

The observation of such unfavorable local minima in trees also provides an explanation why DSA can perform reasonably well on graphs. Considering creating a graph from a tree by connecting the leaf nodes in the tree. For example, we can add edges to the leaf nodes in Fig. 5, making the tree a graph. As a result, the constraints between leaf nodes will be pushed into the internal nodes and consequently be resolved through local variable adjustments.

5.2.2. Communication cost

We have so far focused on DSAs' solution quality without paying any attention to their communication costs. Communication in a sensor network has an inherent 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 (see Algorithm 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, 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. Fig. 6 shows the communication cost on grids with $k = 8$ using 4 colors (left) and 5 colors (right) after 1000 steps. Comparing

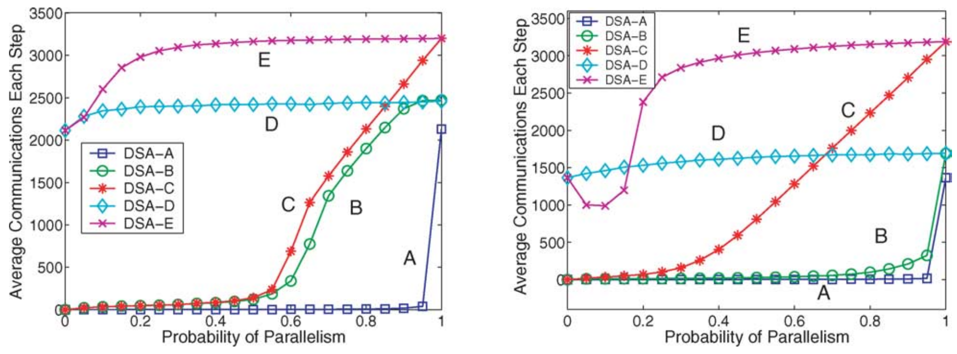


Fig. 6. Communication phase transitions on grids with $k = 8$ using 4 colors (left) and 5 colors (right).

these figures with those in Fig. 2, it is evident that solution quality and communication cost follow almost identical patterns. Furthermore, the communication cost on graphs (not shown here) follows similar patterns as those on grids.

6. Breakout and distributed breakout

Distributed breakout algorithm (DBA) is based on a centralized predecessor. To better understand the distributed algorithm, we start with the centralized version. We then study the strength and weakness of DBA for graph coloring.

6.1. The algorithms

The breakout algorithm [10] is a local search method equipped with an innovative scheme of escaping local minima for CSP. Given a CSP, the algorithm first assigns a weight of one to all constraints. It then picks a value for every variable. If no constraint is violated, the algorithm terminates. Otherwise, it chooses a variable that can reduce the total weight of the unsatisfied constraints if its value is changed. If such a weight-reducing variable-value pair exists, the algorithm changes the value of a chosen variable. The algorithm continues the process of variable selection and value change until no weight-reducing variable can be found. At that point, it reaches a local minimum if a constraint violation still exists. Instead of restarting from another random initial assignment, the algorithm tries to escape from the local minimum by increasing the weights of all violated constraints by one and proceeds as before. This weight change will force the algorithm to alter the values of some variables to satisfy the violated constraints.

Centralized breakout can be extended to distributed breakout algorithm (DBA) [16,18]. Without loss of generality, we assign an agent to a variable, and assume that all agents have unique identifiers. Two agents are *neighbors* if they share a common constraint. An agent communicates only with its neighbors. At each step of DBA, an agent exchanges

Algorithm 2. Sketch of DBA

```
set the local weights of constraints to one
value ← a random value from domain
while (no termination condition met) do
    exchange value with neighbors
    WR ← BestPossibleWeightReduction()
    send WR to neighbors and collect their WRs
    if (WR > 0) then
        if (it has the biggest improvement among neighbors) then
            value ← the value that gives WR
        end if
    else
        if (no neighbor can improve) then
            increase violated constraints' weights by one
        end if
    end if
end while
```

its current variable value with its neighbors, computes the possible weight reduction if it changes its current value, and decides if it should do so. To avoid simultaneous variable changes among neighboring agents, only the agent having the maximal weight reduction has the right to alter its current value. If ties occur, the agents break the ties based on their identifiers. The above process of DBA is sketched in Algorithm 2. For simplicity, we assume each step is synchronized among the agents. This assumption can be lifted by a synchronization mechanism [14].

In the description of [16,18], each agent also maintains a variable, called *my-termination-counter* (MTC), to help detect a possible termination condition. At each step, an agent's MTC records the diameter of a subgraph centered around the agent within which all the agents' constraints are satisfied. For instance, an agent's MTC is zero if one of its neighbors has a violated constraint; it is equal to one when its immediate neighbors have no violation. Therefore, if the diameter of the constraint graph is known to each agent, when an agent's MTC is equal to the known diameter, DBA can terminate with the current agent values as a satisfying solution. However, MTCs may never become equal to the diameter even if a solution exists. There are cases in which the algorithm is not complete in that it cannot guarantee to find a solution even if one exists. Such a worst case depends on the structure of a problem, a topic of the next section. We do not include the MTC here to keep our description simple.

It is worthwhile to point out that the node, or agent, identifiers are not essential to the algorithm. They are only used to establish a priority between two competing agents for tie breaking. As long as such priorities exists, node identifiers are not needed. Nevertheless, such priorities still make DBA less applicable than DSA, since establishing node priorities may not be always feasible in all situations and DSA does not require node priorities at all.

6.2. Completeness and complexity

Distributed algorithms that do not have a global coordination mechanism, are incomplete in that they cannot guarantee to find a solution even if one exists, which is true for the algorithms we consider here. It is also generally difficult to show the completeness of a distributed algorithm and to analyze its complexity. In this section, we consider the completeness and complexity of DBA on a special class of problems, i.e., distributed graph coloring. Here, the complexity is defined as DBA's number of synchronized distributed steps. In one step, value changes at different nodes are allowed while one variable can change its value at most once. We also use the terms variables, nodes and agents interchangeably in our discussion.

6.2.1. Acyclic graphs

First notice that acyclic graphs or trees are 2-colorable. Since graph coloring is in general less constrained with more colors, it is sufficient to consider coloring acyclic graphs with two colors. To simplify our discussion, we first consider chains, which are special acyclic graphs. The results on chains will also serve as a basis for trees.

Chains. We will refer to the combination of variable values and constraint weights as a *problem state*, or *state* for short. A *solution* of a constraint problem is a state with no violated constraint. We say two states are *adjacent* if DBA can move from one state to the other within one step.

Lemma 6.1. *DBA will not visit the same problem state more than once when* 2*-coloring a chain.*

Proof. Assume the opposite, i.e., DBA can visit a state twice in a process as follows, $S_x \to S_y \to \cdots \to S_z \to S_x$. Obviously no constraint weight is allowed to increase at any state on this cycle. Suppose that node x changes its value at state S_x to resolve a conflict C involving x. In the worst case a new conflict at the other side of the node will be created. C is thus "pushed" to the neighbor of x, say y. Two possibilities exist. First, C is resolved at y or another node along the chain, so that no state cycle will form. Second, C returns to x, causing x to change its value back to its previous value. Since nodes are ordered, i.e., they have prioritized identifiers, violations may only move in one direction and C cannot return to x from y without changing a constraint weight. This means that C must move back to x from another path, which contradicts the fact that the structure is a chain. □

Lemma 6.2. *DBA can increase a constraint weight to at most* $\lfloor n/2 \rfloor$ *when coloring a chain of* n *variables using at least two colors.*

Proof. The weight of the first constraint on the left of the chain will never change and thus remain at one, since the left end node can always change its value to satisfy its only constraint. The weight of the second constraint on the left can increase to two at the most. When the weight of the second constraint is two and the second constraint on the left is violated, the second node will always change its value to satisfy the second constraint because it has a higher weight than the first constraint. This will push the violation to the left end node and force it to change its value and thus resolve the conflict. This argument can be inductively applied to the other internal nodes and constraints along the chain. In fact, it can be applied to both ends of the chain. So the maximal possible constraint weight on the chain will be $\lfloor n/2 \rfloor$. □

Immediate corollaries of this lemma are the best and worst arrangements of variable identifiers. In the best case, the end nodes of the chain should be most active, always trying to satisfy the only constraint, and resolving any conflict. Therefore, the end nodes should have the highest priority, followed by their neighbors, and so on to the middle of the chain. The worst case is simply the opposite of the best case; the end nodes are most inactive and have the lowest priority, followed by their neighbors, and so on.

Lemma 6.2 applies to a chain with no restriction on the color a node can take. In fact, we can prove a more general scenario where nodes may be restricted to fixed colors, which may lead to cases where no solution exists.

Theorem 6.1. *DBA terminates in at most* n^2 *steps with a solution, if it exists, or with an answer of no solution, if it does not exist, when coloring a chain of* n *variables using at least two colors.*

Proof. As a chain is always 2-colorable, the combination of the above lemmas gives the result for a chain with nodes of domain sizes at least two. It is possible, however, that no solution exists if some variables have fixed values or colors. In this case, it is easy to create a conflict between two nodes with domain size one, which will never be resolved. As a result, the weights of the constraints between these two nodes will be raised to n. If each agent knows the chain length n, DBA can be terminated when a constraint weight is more than n. (In fact, the chain length can be computed in $\mathrm{O}(n)$ steps as follows. An end node first sends number 1 to its only neighbor. The neighboring node adds one to the number received and then passes the new number to the other neighbor. The number reached at the other end of the chain is the chain length, which can be subsequently disseminated to the rest of the chain. The whole process takes $2n$ steps.) Furthermore, a node needs at most $n-1$ steps to increase a constraint weight if it has to do so. This worst case occurs when a chain contains two variables at two ends of the chain which have the lowest priorities and unity domain sizes so neither of them can change its value. On such a chain, a conflict can be pushed around between the two end nodes many time. Every time a conflict reaches an end node, the node increases the constraint weight to push the conflict back. Since a constraint weight will be no more than n, the result follows. □

A significant implication of these results is a termination condition for DBA for coloring a chain. If DBA does not find a solution in n^2 steps, it can terminate with the conclusion that no solution exists. This new termination condition and DBA's original termination condition of my-termination-counter guarantee DBA to terminate on a chain.

Trees. The key to the proof for coloring chain and tree structures is that no cycle exists in an acyclic graph, so that the same conflict cannot return to a node without increasing a constraint weight.

The arguments on the maximal constraint weight for coloring chains hold for general acyclic graphs or trees. First consider the case that each variable has a domain size at least two. In an acyclic graph, an arbitrary constraint (link) C connects two disjoint acyclic graphs, G_1 and G_2. Assume G_1 and G_2 have n_1 and n_2 nodes, respectively, and $n_1 \leqslant n_2$. Then the maximal possible weight W on C cannot be more than n_1, which is proven inductively as follows. If the node v associated with C is the only node of G_1, then the claim is true since v can always accommodate C. If G_1 is a chain, then the arguments for Lemma 6.2 apply directly and the maximal possible weight of a constraint is the number of links the constraint is away from the end variable of G_1. If v is the only node in G_1 that connects to more than one constraint in G_1, which we call a branching node, then a conflict at C may be pushed into G_1 when the weight of C is greater than the sum of the weights of all constraints in G_1 linked to v, which is at most equal to the number of nodes of G_1. The same arguments equally apply when v is not the only branching node of G_1. Therefore, the maximal constraint weight is bounded by n.

The worst-case complexity can be derived similarly. A worst case occurs when all end variables of an acyclic graph have fixed values, so that a conflict may never be pushed out of the graph. A constraint weight can be bumped up by one after a conflict has traveled from an end node to other end nodes and back, within at most n steps.

Based on these arguments, we have the following result.

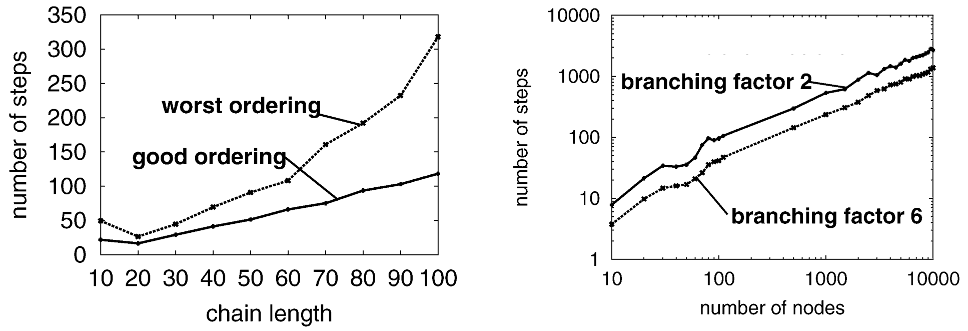


Fig. 7. The number of steps taken by DBA on chains with the best and worst variable identifier arrangements (left) and on trees with worst identifier arrangements (right).

Theorem 6.2. *DBA terminates in at most n^2 steps with either an optimal solution, if it exists, or an answer of no solution, if it does not exist, when coloring an acyclic graph with n nodes with at least two colors.*

The above completeness result can be directly translated to centralized breakout algorithm, leading to its completeness of coloring acyclic graphs as well. Moreover, since each step in DBA is equivalent to n steps in the centralized algorithm, each of which examines a distinct variable, the complexity result on DBA also means that the worst-case complexity of the centralized algorithm is $O(n^3)$. These analytical results reveal the superiority of centralized breakout algorithm and DBA over conventional local search methods of coloring acyclic graphs, including the distributed stochastic algorithm discussed in Section 5, which are not complete even on a chain.

Our experimental results also show that the average number of steps taken by DBA is much smaller than the n^2 upper bound, as shown in Fig. 7. In our experiments, we used different size chains and trees and averaged the results over 100 random trials. We considered the best- and worst-case identifier arrangements for chains (Fig. 7 (left)) and worst-case arrangement for trees (where more active nodes are closer to the centers of the trees) with different branching factors. As the figures show, the average number of steps taken by DBA is near linear for the worst-case identifier arrangement, and the number of steps is linear on trees with a worst-case identifier arrangement (Fig. 7 (right)). Furthermore, for a fixed number of nodes the number of steps decreases inversely when branching factors of the trees increase. In short, DBA is efficient on coloring acyclic graphs.

6.2.2. Cyclic graphs

Unfortunately, DBA is not complete on cyclic constraint graphs. This will include problems of non-binary constraints as they can be converted to problems of binary constraints with cycles. This is also the reason that breakout algorithm is not complete on Boolean satisfiability with three variables per clause [10], which is equivalent to a constraint with three variables.

When there are cycles in a graph, conflicts may walk on these cycles forever. To see this, consider a problem of coloring a ring with an even number of nodes using two colors

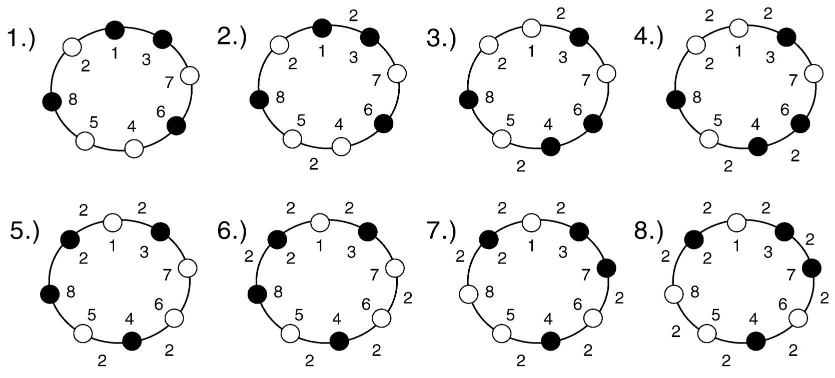


Fig. 8. A worst case of DBA for coloring a ring.

(black and white), as shown in Fig. 8, where the node identifiers and constraint weights are respectively next to nodes and edges. Fig. 8(1) shows a case where two conflicts appear at locations between nodes 1 and 3 and between nodes 4 and 5, that are not adjacent to each other. The weights of the corresponding edges are increased accordingly in Fig. 8(2). As node 1 (node 4) has a higher priority than node 3 (node 5), it changes its value and pushes the conflict one step counter-clockwise in Fig. 8(3). The rest of Fig. 8 depicts the subsequent steps until all constraint weights have been increased to 2. This process can continue forever with the two conflicts moving in the same direction on the chain at the same speed, chasing each other endlessly and making DBA incomplete.

6.3. Stochastic variations

A lesson that can be learned from the above worst-case scenario is that conflicts should not move at the same speed. We thus introduce randomness to alter the speeds of possible conflict movements on cycles of a graph. This stochastic feature may increase DBA's chances of finding a solution possibly with a penalty on convergence to solution for some cases.

6.3.1. DBA(wp) *and* DBA(sp)

We can add randomness to DBA in two ways. In the first, we use a probability for tie breaking. The algorithm will proceed as before, except that when two neighboring variables have the same improvement for the next step, they will change their values probabilistically. This means that both variables may change or not change, or just one of them. We call this variation weak probabilistic DBA, denoted as DBA(wp).

In the second method, which was inspired by the distributed stochastic algorithm [2,3, 20], a variable will change if it has the best improvement among its neighbors. However, when it can improve but the improvement is not the best among its neighbors, it will change based on a probability. This variation is more active than DBA and the weak probabilistic variation. We thus call it strong probabilistic DBA, DBA(sp) for short.

One favorable feature of these variants is that no node identifiers or priorities are needed, which may be important for some applications where node identifiers or priorities across

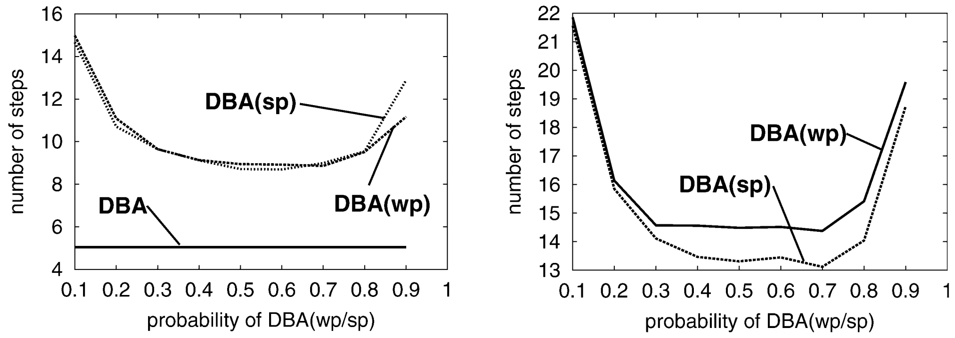


Fig. 9. Steps taken by DBA and variants on the example of Fig. 8 with random initial assignments (left) and the specific worst-case assignment of Fig. 8 (right).

the whole network is expensive to compute. Moreover, these variants give two families of variations to DBA, depending on the probabilities used. It will be interesting to see how they vary under different parameters, the topic that we consider next.

6.3.2. DBA(wp) *versus* DBA(sp)

We first study the two variants on the example of coloring an 8-node ring of Fig. 8. In the first set of tests, node identifiers and initial colors were randomly generated and 10,000 trials were tested. DBA was unable to terminate on 15% of the total trials after more than 100,000 steps,[3] while on the other 85% of the trials DBA found a solution after 5 steps on average as shown in Fig. 9 (left). In contrast, DBA(wp) and DBA(sp) always found solutions but required almost twice as many steps on average with the best probability around 0.6 to 0.7.

In the second set of tests, we used the exact worst-case initial assignment as shown in Fig. 8. As expected, DBA failed to terminate. DBA(wp) and DBA(sp) found all solutions on 1000 trials. Since they are stochastic, each trial may run a different number of steps. The average number of steps under different probability is shown in Fig. 9 (right).

Comparing the results in Fig. 9 (left) and Fig. 9 (right), we see that the peculiar worst-case initial assignment made DBA(wp) and DBA(sp) take extra steps to break out from deadlock states, resulting in longer executions.

Next we studied these two families of variants on grids, graphs and trees. We considered coloring these structures using 2 colors. For grids, we consider 20×20, 40×40, and 60×60 grids with connectivities of $k = 4$ and $k = 8$. The results of 20×20 grids with $k = 4$ are shown in Fig. 10, averaged over 2000 trials. As the figures show, the higher the probability, the better DBA(wp)'s performance. For DBA(sp), $p = 0.5$ is the best probability.

We generated 2000 random graphs with 400 nodes with an average connectivity per node equal to $k = 4$ and $k = 8$ by adding, respectively, 1600 and 3200 edges to randomly picked pairs of unconnected nodes. These two graphs were used to make a correspondence to the grid structures of $k = 4$ and $k = 8$ considered previously, except that both random

[3] Our additional tests also show that DBA's failure rate decreases as the ring size increases.

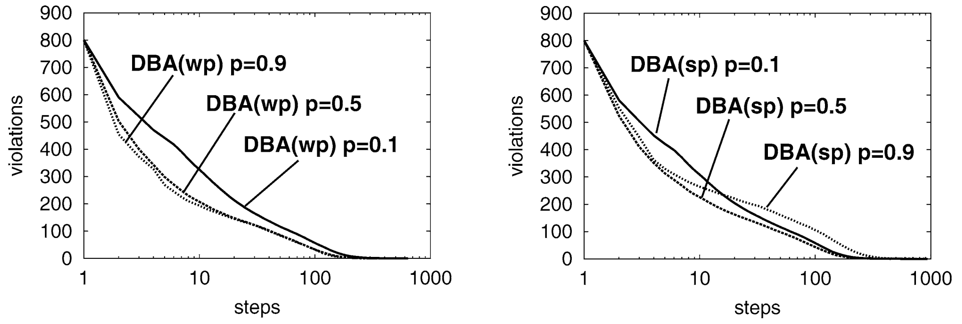


Fig. 10. DBA(wp) and DBA(sp) on grid 20 × 20 and $k = 4$.

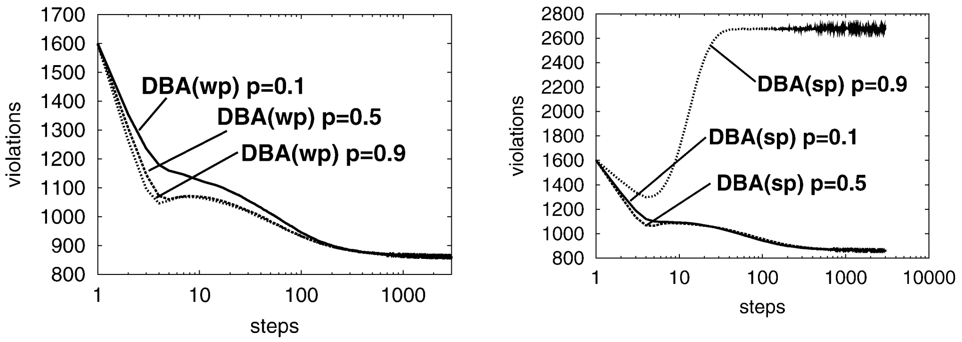


Fig. 11. DBA(wp) (left) and DBA(sp) (right) on graph with 400 nodes and $k = 8$.

graphs may not be two-colorable. All algorithms were applied to the same set of graphs for a meaningful comparison. Fig. 11 shows the results on graphs with $k = 8$. There is no significant difference within the DBA(wp) family. However, DBA(sp) with large probabilities can significantly degrade to very poor performance, exhibiting a phenomenon similar to phase transitions. Since DBA(sp) with high probability acts more like the distributed stochastic algorithm [2,3,20], the degradation of DBA(sp) observed here is in line with the results of DSA with high parallelism, as discussed in Section 5.2.

We also considered DBA(wp) and DBA(sp) on random trees with various depths and branching factors. Due to space limitations, we do not include detailed experimental results here, but give a brief summary. As expected, these algorithms all found optimal solutions for all 10,000 2-coloring instances. Within DBA(wp) family, there is no significant difference. However, DBA(sp) with a high probability has a poor anytime performance.

Combining all the results on the constraint structures we considered, DBA(sp) appears to be a poor algorithm in some cases, especially when its probability is very high.

6.3.3. DBA(wp) *and* DBA(sp) *versus* DBA

An unsettled issue so far is how DBA(wp) and DBA(sp) compare with DBA. Here we used the best parameters for these two variants from the previous tests and compared them directly with DBA. We averaged the results over the same sets of problem instances we

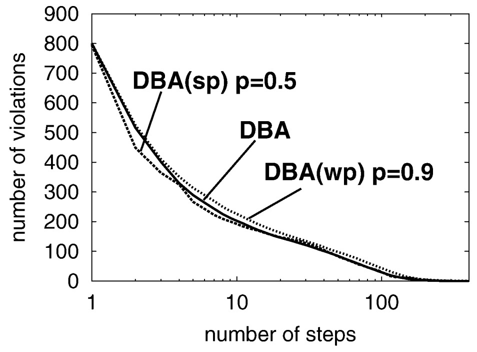


Fig. 12. DBA and random DBAs on grid 20 × 20 and $k = 4$.

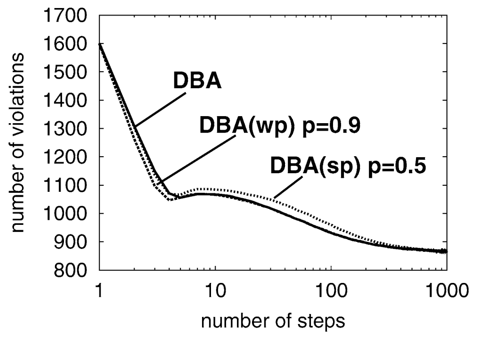


Fig. 13. DBA and random DBAs on graph with 400 nodes and $k = 8$.

used in Section 6.3.2. Figs. 12, 13 and 14 show the experimental results on grids, random graphs and trees, respectively. With their best parameters, DBA(wp) and DBA(sp) appear to be comparable to DBA. Furthermore, as discussed earlier, DBA(wp) and DBA(sp) increase the probability of convergence to optimal solutions. DBA(wp), in particular, is a better alternative in many cases if its probability is chosen carefully. Stochastic features do not seem to impair DBA's anytime performance on many problem structures and help overcome the problem of incompleteness of DBA on graphs with cycles. Therefore, DBA(wp) should be used in place of DBA.

7. Applications and comparative analysis

We now directly compare DBA and DSA on graph coloring problems generated from the scan scheduling problem discussed in Section 2.1. In our experiments, we set the sensing radius of a sensor to one unit, and used a square of 10 × 10 units as the area to be monitored. The number of sensing sectors was set to three to match our hardware system. We randomly and uniformly placed a fixed number of sensors with arbitrary orientations in the square. We then converted these problems to graph coloring problems as described in Section 3. We experimented with different values of maximum allowed colors T, which

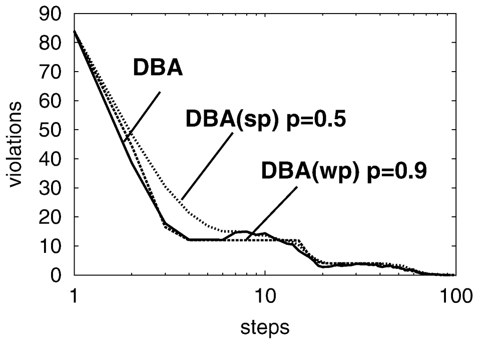


Fig. 14. DBA and random DBAs on tree with depth $d = 4$ and branching factor $k = 4$.

correspond to the cycle steps for scanning, and different sensor activation ratios α, which determine how often the sensors will be active within a scanning cycle. In the following, we report the results using $T = 6$ and $T = 18$ with $\alpha = 2/3$.

For DSA, we changed its probability p of parallel executions from 0.1 to 0.99, with an increment of 0.01. We used 100 instances for each p. We evaluated the performance of DSA and DBA when they have reached relatively stable states. Specifically, we ran DSA and DBA to the point where their performance does not change significantly from one step to the next. On all network sizes we considered, these algorithms' performances seem to stabilize after more than 250 steps. Similar results have been observed after 1024, 2048 and more steps. In the rest of this paper, we report the results at 256 steps.

7.1. Solution quality in terms of network sizes

Since one of our ultimate objectives is to choose DBA or DSA to solve our distributed scan scheduling problem, we need to investigate the relationship between the quality of the schedules found by these two algorithms and the properties of the underlying networks. To this end, we experimentally compared DBA and DSA on graph coloring problems produced from sensor networks of various sizes. We changed the density of graph coloring graphs by changing the number of sensors N. The solution quality is the total weight of violated soft constraints normalized by the total weight of soft constraints, measured at 256 steps of the algorithms' executions when their performances are relatively stable.

We run DBA and DSA with four different representative probabilities p of parallel executions, 0.1, 0.4, 0.7 and 0.9. We varied the density or number of sensors and compared the quality of the colorings that DSA and DBA produced. We changed the number of sensors from 25 to 100 with an increment of 5 sensors, and from 100 to 500 with an increment of 25 sensors. We averaged the results over 100 random problem instances for each fixed number of sensors. Fig. 15 shows the result on $T = 6$. The horizontal axes in the figures are the numbers of sensors, and the vertical axes are the normalized solution quality after 256 steps. Longer executions, such as 512 and 1024 steps, exhibit almost identical results. Fig. 15(a) shows the result in the whole range of 25 to 500 sensors and Fig. 15(b) expands the results of Fig. 15(a) in the range of 25 to 100 sensors.

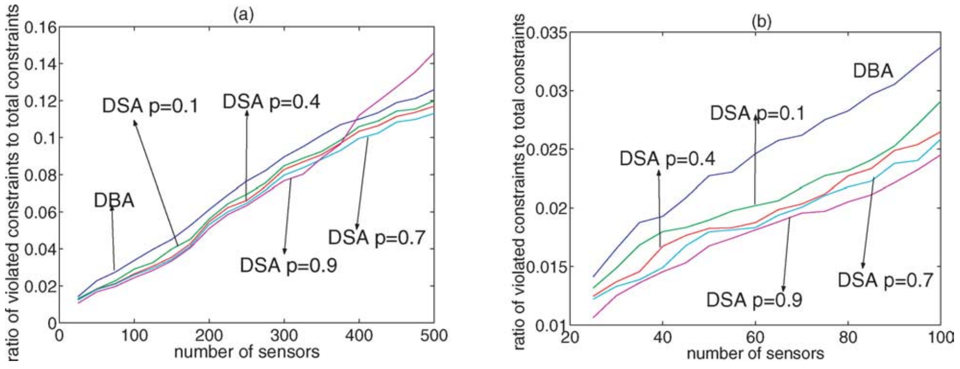


Fig. 15. DSA vs. DBA in terms of number of sensors, $T = 6$.

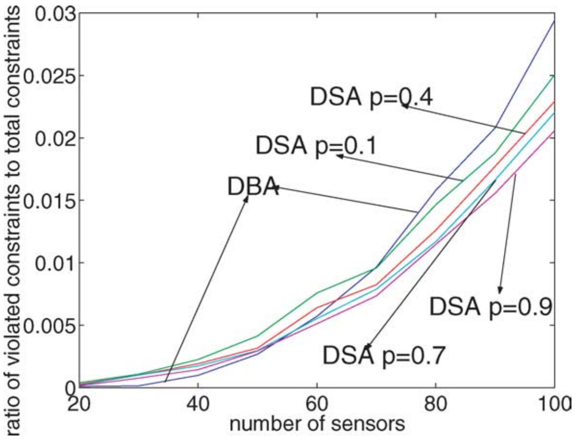


Fig. 16. DSA vs. DBA in terms of number of sensors, $T = 18$.

As analyzed in Section 6, DBA may perform better than DSA on underconstrained problems, especially acyclic graphs. An underconstrained scan scheduling problem may be created when more colors are available. Indeed, when we increased the number of allowed colors (targeting schedule cycle length) to eighteen ($T = 18$), DBA outperformed DSA on sparse networks with less than 50 sensors. This result is shown in Fig. 16, where each data point is averaged over 100 trials.

Based on the experimental results, we can reach three conclusions. First, DBA typically performs worse than DSA when its degree of parallelism is not too high in the range of 25 to 500 sensors. Second, when the sensor density increases, the performance of DSA may degenerate, especially if its degree of parallelism p is high. Particularly, on over-constrained networks, a high parallelism may lead to a significantly performance degradation. For instance, the performance of DSA with $p = 0.9$ degrades quickly when there are more than 400 sensors (Fig. 15). The degenerated performance of DSA with a large p is mainly due to its phase-transition behavior revealed in the previous section. When the sensor density increases, more constraints will be introduced into the constraints of the scan scheduling problem, so that the problem becomes overconstrained. As indicated in the phase-transition section, DSA's phase-transition behavior appears sooner when overall constraints are tighter. Third, in the underconstrained region, a higher degree of parallelism is preferred to a lower degree.

7.2. Anytime performance

An important feature of our targeting sensor network for object detection is real-time response. High real-time performance is important, especially for systems with limited computation and communication resources in which it may be disastrous to wait for the systems to reach stable or equilibrium states. This is particularly true for our sensor-based system for object detection and mobile object tracking. Therefore, the algorithm for the distributed scan scheduling must have anytime property, i.e., the algorithm can be stopped at anytime during its execution and the set of the then current variable assignments of individual agents can be returned as a solution. Fortunately, DBA and DSA can both be used for this purpose because the hard constraints internal to individual sensors (a sensor cannot scan its two sectors at the same time) are always maintained.

In the rest of this section, we directly compare DBA and DSA as anytime algorithms. Note that the conventional notion of anytime algorithm requires an anytime algorithm to return the best solution found before it was terminated. Here, we extend this conventional notion to allow these algorithms to return the current variable assignments as a solution if they are terminated at particular time. A critical issue is then the quality of the current solutions these algorithms can produce during the whole process of their executions. To investigate this issue, we resort to experimental analysis.

In our experiments, we first considered dense networks with $N = 500$ and $N = 300$ sensors. We used the same set of experimental conditions and parameters as in the previous sections, i.e., sensors have three sectors and are randomly and uniformly placed on a 10×10 grid, with results averaged over 100 trials. Based on the phase-transition results in Section 5.2, DSA performs the best with $p = 0.9$ and $p = 0.78$ for the networks of $N = 500$ and $N = 300$ nodes, respectively. We used these parameters in our experiments.

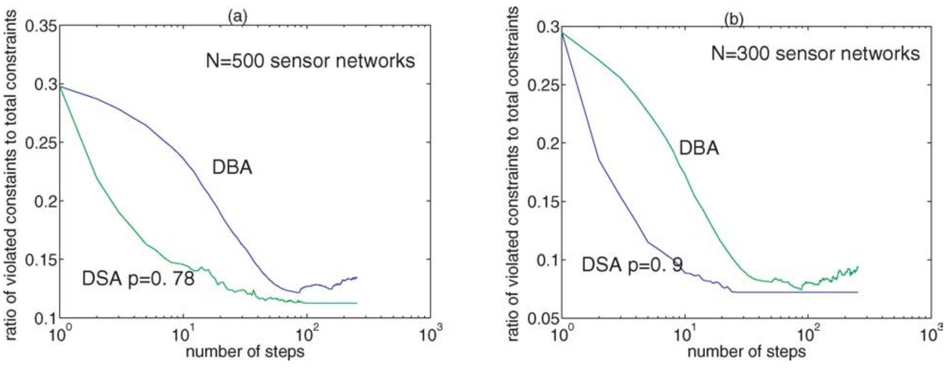


Fig. 17. Anytime performance of DSA and DBA in dense sensor networks, $T = 6$.

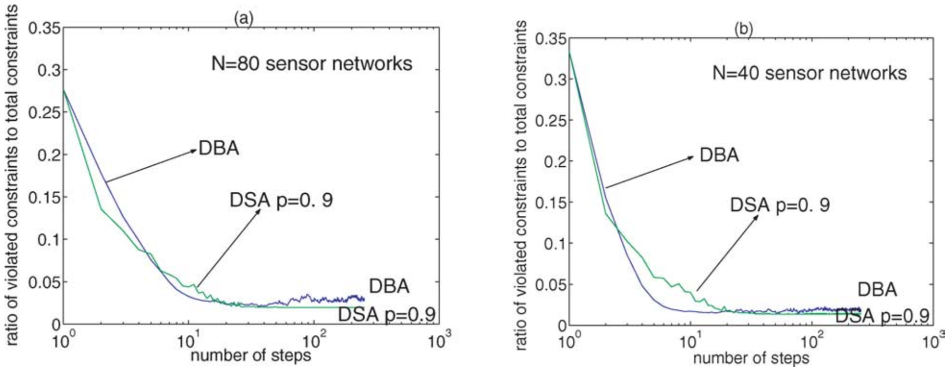


Fig. 18. Anytime performance of DSA and DBA in sparse sensor networks, $T = 6$.

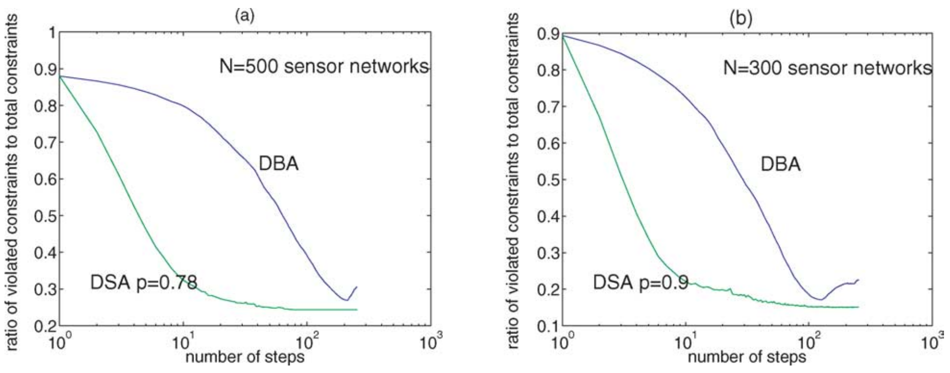


Fig. 19. Anytime performance of DSA and DBA in dense sensor networks, $T = 18$.

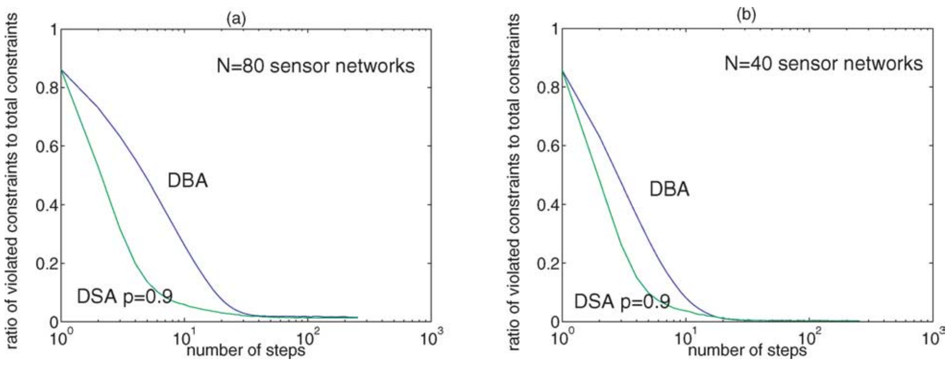


Fig. 20. Anytime performance of DSA and DBA in sparse sensor networks, $T = 18$.

The experimental results are in Fig. 17. As the results show, DSA performs much better than DBA in both anytime performance and final solution quality.

We now consider sparse sensor networks, using networks with $N = 80$ and $N = 40$ sensors as representatives. As discussed earlier, a higher degree of parallelism should be used on sparse graphs, we thus used $p = 0.9$ for our two sparse networks. The results, averaged over 100 trials, are in Fig. 18. Clearly, DBA and DSA exhibits similar performance, with DSA being able to produce slightly better solutions at the end.

To complete our analysis, we also compared DSA and DBA on the same sets of instances, but with 18 available colors ($T = 18$). The results on dense and sparse sensor networks are included in Figs. 19 and 20, respectively. Interestingly, DBA's anytime performance degenerates, compared to that using $T = 6$.

In summary, as far as solution quality (anytime and final solutions) is concerned, our experimental results indicate that DSA should be adopted for the distributed scan scheduling problem.

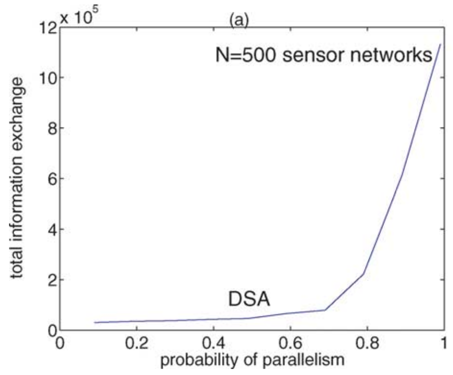
7.3. Communication cost

As mentioned earlier, communication in a sensor network has an inherent delay and could be unreliable in most situations. Therefore, a good distributed algorithm should require a small number of message exchanges.

In each step of DBA, an agent announces its best possible conflict reduction to its neighbors and receives from the neighbors their possible weight reductions. Thus, the number of messages sent and received by an agent in each step of DBA is no less than the number of its neighbors, and the total number of messages exchanged in each step is more than a constant for a given network.

In contrast, an agent in DSA may not send a message in a step if it does not have to change its value. In an extreme case, an agent will not change its value if it is at an local minima. If solution quality of DSA improves over time, its communication cost will reduce as well. In principle, the communication cost of DSA is correlated to its solution quality. The better the current solution, the less the number of messages. In addition, the communication cost is also related to the degree of parallel executions of the agents. The higher the parallel probability p is, the higher the communication cost will be. As shown in Section 5, the communication cost of DSA goes hand-in-hand with its solution quality and also experiences a similar phase-transition or threshold behavior on regular coloring problems. Fig. 21 shows the phase-transition behavior of DSA's communication cost on the $N = 500$ and $N = 300$ sensor networks using $T = 6$ that we studied before. Here we considered the accumulative communication cost of all 256 steps. This result indicates that the degree of parallelism must be controlled properly in order to make DSA effective. Similar phase-transition patterns have been observed when we use $T = 18$.

We now compare DSA and DBA in terms of communication cost. Fig. 22 shows the results evaluating DBA and DSA with probability $p = 0.78$ on $N = 500$ networks using $T = 6$ and $T = 18$, averaged over 100 trials. The figures plot the average numbers of messages exchanged in DSA and DBA at a particular step. Clearly, DSA has a significant advantage over DBA on communication cost. The large difference on communication cost between DSA and DBA will have a significant implication on how these two algorithms can be used in real sensor networks, especially when the sensors are connected through delayed, unreliable and noisy wireless communication. For our particular application and



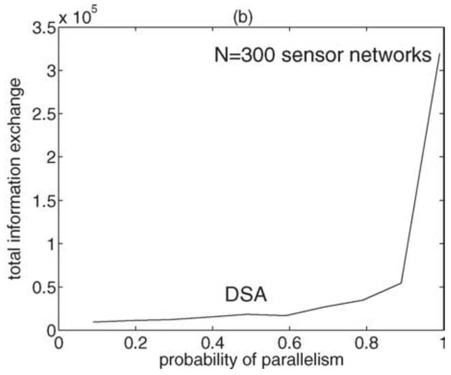
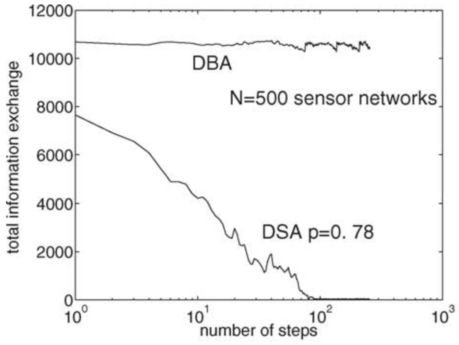


Fig. 21. Communication-cost phase transitions of DSA on scan scheduling, $T = 6$.



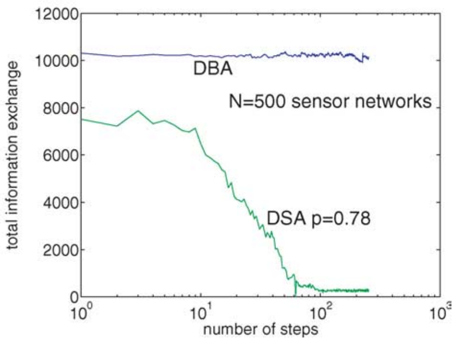


Fig. 22. Communication cost of DSA and DBA, $T = 6$ (left) and $T = 18$ (right).

system where communication was carried out by radio frequencies, DBA's high communication cost makes it noncompetitive.

In summary, in terms of solution quality and communication cost, DSA is preferable over DBA for our distributed scan scheduling if DSA's degree of parallelism is properly controlled.

7.4. Solving scheduling problem

Based on the results from Sections 5.2–7.3, we now apply DSA and DBA to dealing with two related problems at the same time, finding the shortest scan cycle length T and obtaining a good schedule given the shortest cycle length T.

To this end, we run DSA and DBA in iterations, starting with an initially large T. T is reduced after each iteration. Given a T in an iteration, DSA or DBA searches for a schedule of a quality better than a predefined threshold Q. The iteration stops whenever such a schedule is found within a fixed number of steps, and a new iteration may start with a smaller T.

In our simulation, we checked the quality of the current schedule after each simulated step. As soon as the quality of the current schedule exceeds the given threshold Q, we

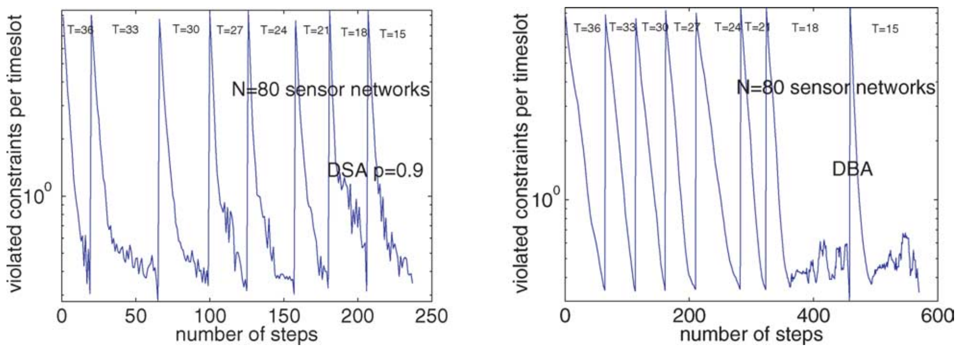


Fig. 23. Finding best possible schedule using DSA (left) and DBA (right), $N = 80$.

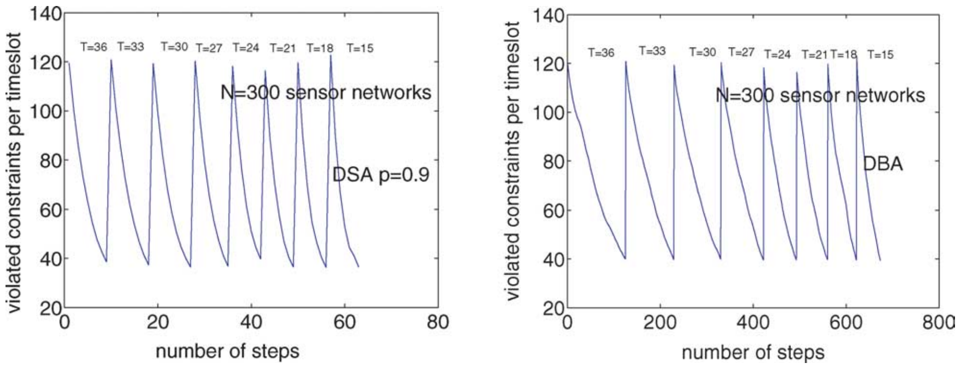


Fig. 24. Finding best possible schedule using DSA (left) and DBA (right), $N = 300$.

terminate the current iteration. This is equivalent to having an agent compute the global state of a distributed system, a method infeasible for our completely distributed system. We use this mechanism here simply to evaluate the performance of DSA and DBA.

Figs. 23 and 24 show the results on two networks, one with $N = 80$ sensors and the other with $N = 300$. In our experiments, we fixed the sensor activation ratio at $\alpha = 2/3$, used initial $T = 36$, and reduced T by three after each iteration, which ran a maximum of 256 steps. The threshold for schedule quality was set to $Q = 0.01$ for $N = 80$ and $Q = 40$ for $N = 300$. As the results show, DSA is superior to DBA. On the $N = 80$ network (Fig. 23), DSA finds a targeting schedule of length $T = 15$ in 242 steps, while DBA needs 588 steps. On the $N = 300$ network (Fig. 24), DSA takes 64 steps, while DBA uses 691 steps, which is an order of magnitude difference.

In summary, our results clearly show that DSA is superior to DBA on the distributed scan scheduling problem. If communication cost is also a concern, DSA is definitely the algorithm of choice for the problem. In addition, if a solution of no conflict is required, a large number of T should be used and DBA may be applied to such underconstrained networks for finding a desirable solution.

8. Related work and discussions

The basic idea of distributed stochastic search must have been around for some time. A similar idea was used by Pearl in distributed belief update [11]. The idea was directly used for distributed graph coloring in [2,3]. DSA-B considered here is the same as CFP in [3]. However, [2,3] failed to reveal phase transitions discussed in this paper. The idea was also studied using spin glasses models [8] where phase transitions were characterized. Phase transitions in distributed constraint problem solving was also reported in [6].

This research extends the existing work of DSA 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 DSA's phase transition behavior exists in various problems and under many different problem structures and it persists 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.

Other related algorithms include complete algorithms for DisCSP, such as the asynchronous weak-commitment (AWC) search algorithm [16,17], and for DisCOP, such as the Adopt algorithm [9]. These complete algorithms are important for distributed constraint solving. Comparing to DBA and DSA, however, they require much longer running time and usually have worse anytime performance, making them inferior for real-time applications where optimal solutions may be too costly to obtain, may constantly change or may not be necessary. In addition, these complete algorithms require a sufficiently large amount of memory to record the states (agent views) that an agent has visited in order to avoid revisiting a state multiple times so as to make the algorithms converge to a solution if it exists. In contrast, DSA and DBA are able to reach near optimal solutions quickly without additional memory. This feature, along with their good anytime performance, made DSA and DBA attractive to applications in sensor networks where memory is a crucially limited resource. In addition, on coloring acyclic graphs, DBA is complete and has a low polynomial complexity, making it desirable for finding optimal solutions in such a case.

DSA differs from DBA, AWC and Adopt by the notion of uniformness. A distributed algorithm is called uniform if all nodes execute the same procedure and two nodes do not differ from each other [14]. Therefore, DSA is a uniform algorithm since the nodes do not have identifications and they all execute the same set of instructions. However, DBA, AWC and Adopt are not uniform because the nodes in these algorithms need to have identifications to differ from one another and to set priorities to decide what to execute next.

We need to emphasize that the notion of uniformness for distributed algorithms has a practical importance for applications using sensor networks. In a typical application using sensor networks, sensors may have to dynamically organize to form a system. The use of identifications and of priorities among nodes (sensors) will introduce prohibitive barriers on what systems a set of sensors and a given placement can form. The fact that DSA is a uniform algorithm further supports the conclusion from the comparison results in this paper that DSA is preferable to DBA, especially on overconstrained problems. It is also worth mentioning that the stochastic variations to DBA in Section 6.3 are able to relieve the requirement of node identities and priorities. Therefore, stochastic DBA algorithms, DBA(wp) in particular, should be applied when solving acyclic constraint problems.

9. Conclusions

We were motivated in this research to apply the framework of multiagent systems and the techniques of distributed constraint problem solving to resource bounded, anytime, distributed constraint problems in sensor networks. Our specific applications include the detection of mobile objects and the detection of material damage in real time using distributed sensors and actuators. We first formulated these problems as distributed graph coloring problems with the objective of minimizing the number of violated constraints.

To cope with limited resources and to meet the restricted requirement of anytime performance, we were interested in those distributed algorithms that have low-overhead on memory and computation for solving distributed constraint optimization problems. We focused particularly on the distributed stochastic algorithm (DSA) [2,3,8,11] and the distributed breakout algorithm (DBA) [10,16,18], two existing distributed algorithms that fit into the category of low-overhead distributed algorithms. We analyzed and compared DSA and DBA on distributed graph coloring problems that were generated from our distributed scheduling problems in sensor networks. We specifically investigated the relationship among the degree of parallel executions, problem constrainedness, and DSA's behavior and performance. We showed that DSA exhibits a threshold behavior similar to phase transitions in which its performance, in terms of both solution quality and communication cost, degrades abruptly and dramatically when the degree of agents' parallel execution increases beyond a critical point. We also studied the completeness and complexity of DBA on distributed graph coloring problems, showing that DBA is complete and has low polynomial complexity on coloring acyclic graphs. However, DBA is not complete in general. We also introduced randomization schemes to DBA to improve its worst case performance. Finally, we directly compared DSA and DBA on our application problems of distributed scheduling problems in sensor networks. We showed that if controlled properly, DSA is significantly superior to DBA, finding better solutions with less computational cost and communication overhead. For distributed scheduling problems such as the ones considered in this paper, DSA is the algorithm of choice.

Acknowledgements

This research was supported in part by US National Science Foundation Grants IIS-0196057 and ITR/EIA-0113618, and in part by US Defense Research Projects Agency and Air Force Research Laboratory, Air Force Material Command, USAF, under Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. The views and conclusions herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Government. Thanks to

Stephen Fitzpatrick for many discussions related to distributed stochastic search, and to Sharlee Climer for a critical reading of a manuscript. Thanks also to the anonymous reviewers to this paper and [19–22] for comments and suggestions that improved the quality of the research and presentation.

References

[1] P. Cheeseman, B. Kanefsky, W.M. Taylor, Where the really hard problems are, in: Proc. 12th Internat. Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, 1991, pp. 331–337.

[2] M. Fabiunke, Parallel distributed constraint satisfaction, in: Proc. Internat. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA-99), 1999, pp. 1585–1591.

[3] S. Fitzpatrick, 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, 2001, pp. 49–64.

[4] S. Fitzpatrick, L. Meertens, Experiments on dense graphs with a stochastic, peer-to-peer colorer, in: Proc. AAAI-02 Workshop on Probabilistic Approaches in Search, 2002, pp. 24–28.

[5] J. Hill, D. Culler, Mica: a wireless platform for deeply embedded networks, IEEE Micro. 22 (6) (2002) 12–24.

[6] K. Hirayama, M. Yokoo, K. Sycara, The phase transition in distributed constraint satisfaction problems: First results, in: Proc. Internat. Workshop on Distributed Constraint Satisfaction, 2000.

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

[8] W.G. Macready, A.G. Siapas, S.A. Kauffman, Criticality and parallelism in combinatorial optimization, Science 271 (1996) 56–59.

[9] P.J. Modi, W-M. Shen, M. Tambe, M. Yokoo, An asynchronous complete method for distributed constraint optimization, in: Proc. 2nd Internat. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-03), 2003, pp. 161–168.

[10] P. Morris, The breakout method for escaping from local minima, in: Proc. 11th National Conference on Artificial Intelligence (AAAI-93), Washington, DC, 1993, pp. 40–45.

[11] J. Pearl, Evidential reasoning using stochastic simulation of causal models, Artificial Intelligence 32 (1987) 245–257.

[12] H. Reichl, Overview and development trends in the field of MEMS packaging, Invited talk given at 14th Internat. Conf. on Micro Electro Mechanical Systems, January 21–25, 2001, Switzerland.

[13] M. Takeda, Applications of MEMS to industrial inspection, Invited talk, 14th Internat. Conf. on Micro Electro Mechanical Systems, January 21–25, 2001.

[14] G. Tel, Introduction to Distributed Algorithms, second ed., Cambridge University Press, Cambridge, 2000.

[15] G. Xing, A performance-driven framework for customizing CSP middleware support, Master's thesis, Washington University in St. Louis, Saint Louis, Missouri, USA, April 2003.

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

[17] M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara, The distributed constraint satisfaction problem: formalization and algorithms, IEEE Trans. Pattern Anal. Machine Intell. 10 (5) (1998) 673–685.

[18] M. Yokoo, K. Hirayama, Distributed breakout algorithm for solving distributed constraint satisfaction problems, in: Proc. 2nd Internat. Conference on Multi-Agent Systems (ICMAS-96), 1996, pp. 401–408.

[19] W. Zhang, Z. Deng, G. Wang, L. Wittenburg, Z. Xing, Distribute problem solving in sensor networks, in: Proc. 1st Internat. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-02), 2002, pp. 988–989.

[20] W. Zhang, G. Wang, L. Wittenburg, Distributed stochastic search for distributed constraint satisfaction and optimization: parallelism, phase transitions and performance, in: Proc. AAAI-02 Workshop on Probabilistic Approaches in Search, 2002, pp. 53–59.

[21] W. Zhang, L. Wittenburg, Distributed breakout revisited, in: Proc. 18th National Conference on Artificial Intelligence (AAAI-02), Edmonton, AB, 2002, pp. 352–357.

[22] W. Zhang, Z. Xing, G. Wang, L. Wittenburg, An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks, in: Proc. 2nd Internat. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-03), 2003, pp. 185–192.

[23] J. Zhao, R. Govindan, D. Estrin, Computing aggregates for monitoring wireless sensor networks, in: Proc. 1st IEEE Internat. Workshop on Sensor Network Protocols and Applications, 2003.