

## Chapter 1

# A COMPARATIVE STUDY OF DISTRIBUTED CONSTRAINT ALGORITHMS WITH APPLICATIONS TO PROBLEMS IN SENSOR NETWORKS

Weixiong Zhang<sup>1</sup>, Guandong Wang<sup>1</sup>, Zhao Xing<sup>1</sup>, Lars Wittenburg<sup>1</sup>

<sup>1</sup>*Department of Computer Science and Engineering  
Washington University in St. Louis  
St. Louis, MO 63130, USA  
zhang@cse.wustl.edu*

### **Abstract**

This research is motivated by a distributed scheduling problem in distributed sensor networks, in which computational resources are scarce. To cope with limited computational resources and restricted real-time requirement, it is imperative to apply distributed algorithms that have low overhead on resource requirement and high anytime performance. In this paper, we study distributed stochastic algorithm (DSA) and distributed breakout algorithm (DBA), two distributed algorithms developed earlier for distributed constraint satisfaction problems. We experimentally investigate their properties and compare their performance using our distributed scheduling problem as a benchmark. We first formulate the scheduling problem as a distributed multi-coloring problem. We then experimentally show that the solution quality and communication cost of DSA exhibit phase-transition or threshold behavior, in that the performance degenerates abruptly and dramatically when the degree of parallel executions of distributed agents increases beyond some critical value. The results show that when controlled properly, DSA is superior to DBA, having better or competitive solution quality and significantly smaller communication cost than DBA. Therefore, DSA is the choice of algorithm for our distributed scan scheduling problem.

## 1. Introduction

We have been developing a large-scale, wireless sensor network controlled by a distributed multi-agent system. We place an agent atop of a sensor, which runs on a battery and has, among other things, 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 do on its own. A typical application of our system is mobile object tracking in dynamic and real-time environments, where at least three sensors are required to sense at the same time in order to triangulate the location of a target.

Before the sensors can start to track a mobile object, they need to detect the object when it first enters the overall monitored area. The sensors have to scan the areas that they cover. The sensing actions of the sensors must be scheduled in such a way that new objects can be detected as quickly as possible with as little energy usage as possible. As to be described in the next section, the scan scheduling problem can be cast as a distributed constraint satisfaction problem (CSP) or constraint optimization problem (COP).

Due to the real-time nature of the application and limited computation and communication resources on the physical devices, the key to large-scale, real-time sensor/agent networks is the mechanism that can enable agents to make viable distributed decisions in restricted time with limited computation and communication resources. In this research, we are particularly interested in *low-overhead methods* for solving distributed CSPs/COPs. By low-overhead methods we mean those in which limited communication among agents is needed and agents make their decisions without global knowledge of the system and without knowing the actions of the other agents.

In this paper, we investigate distributed stochastic algorithm (DSA) [Fabiunke, 1999; Fitzpatrick and Meertens, 2001; Zhang et al., 2002] and distributed breakout algorithm (DBA) [Morris, 1993; Yokoo, 2001; Yokoo and Hirayama, 1996; Zhang and Wittenburg, 2002], two existing algorithms for distributed constraint problems, in the context of solving our scan scheduling problem. We consider these algorithms as low-overhead methods because they indeed require limited communications and use simple mechanisms to search, distributedly, for global solutions. The main differences between these two approaches are that DBA introduces priorities for conflict resolution among agents' actions, while DSA uses randomness to try to avoid conflicts.

We apply DSA and DBA to our distributed scan scheduling problem and evaluate their performance. In particular, we investigate their characteristics on this distributed problem, and compare their performance using sensor networks of various sizes. The objective is to identify the right algorithm and its appropriate parameters for our application.

A detailed account of the scan scheduling problem is given in Section 1.2. In Section 1.3, we formulate the problem as a weighted multi-coloring problem, in which a node may have multiple colors. We then briefly describe DBA and DSA in Section 1.4. We also discuss how to extend DBA to handle weighted constraints. As reported in our previous research, DSA exhibits phase-transition or threshold behavior if its parameter is not controlled properly when applied to regular graph coloring problems. To choose the right parameters to avoid DSA's degraded performance on our scan scheduling problem, we reinvestigate the phase-transition issue on multi-coloring problems in Section 1.5. We compare the performance of DSA and DBA at their plausible equilibrium states as well as their anytime performance in Section 1.6. We evaluate their communication cost at Section 1.7. In Section 1.8, we evaluate DSA and DBA for finding the shortest scan schedule of quality exceeding a threshold. Finally, we conclude in Section 1.9.

## 2. Distributed Scan Scheduling

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 are developing such an object detecting and tracking system using MEMS sensors, each of which is operated 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. However, 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 signals. 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 energy. To save energy, a sensor may turn itself off occasionally or periodically if doing so does not reduce system performance.

The overall system is operated to achieve two conflicting goals, to detect foreign objects as many and quickly as possible and, at the same time, 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 two conflicting goals.

One of our goals 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 atop 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 local information including its neighboring sensors or agents. These agents also try to avoid unnecessary communication so as to minimize system response time. Therefore, complex coordination and reasoning methods are not suitable.

We set forth to apply simple, low-overhead methods for solving this distributed scan scheduling problem. We consider two distributed algorithms, distributed breakout and distributed stochastic search, that were developed earlier to solve distributed constraint satisfaction problems. Our objectives are twofold. First, we want to understand how these two algorithms compare with each other on a real-world application such as our distributed scan scheduling problem. Second, we want to identify the right application conditions and parameters of the algorithm we can apply based on the evaluation.

### 3. Model in Multiple-coloring

Our first step to address the distributed scan scheduling problem is to model it as a distributed constraint satisfaction or optimization problem. Indeed, the problem can be captured as a distributed multi-coloring problem.

To detect objects quickly, 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 objective 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, an assumption that can be lifted by a synchronization method [Tel, 2000]. (In fact, the algorithms we will consider shortly, DSA and DBA, are synchronous. It has been observed that under certain conditions asynchronous actions may actually improve the performance of a distributed algorithm [Fitzpatrick and Meertens, 2001].) Therefore, the problem can be viewed as coloring a weighted graph so that the total weight of violated constraints is minimized. Here, a node corresponds to a sensor or agent, a link between two nodes represents the constraint of a shared region between two agents, and the weight measures the area of a common region. The larger a shared region is, the more energy will be wasted if two sensors scan the shared region at the same time. Moreover, each color corresponds to a time slot in which a sector is scanned and the total number of colors represents the scanning cycle length or the number of time slots required to scan each of the sectors of all sensors.

Furthermore, a node must have at least one color so that the corresponding sector is scanned at least once, and may also have multiple colors so that the sector may be sensed multiple times to increase system's object detection rate.

We further consider a sensor as an hypernode of three nodes, each of which corresponds to a sector of the sensor. Within an hypernode, two nodes cannot share a color since two sectors cannot be scanned at the same time. In other words, the colors assigned to the nodes within an hypernode are mutually exclusive. This is a hard constraint that cannot be violated. Furthermore, not all available colors must be used as a sensor may sometimes turn itself off. A node within a hypernode may also be constrained by a node of another hypernode if the two corresponding sectors share a common region. This constraint may be violated, resulting in wasted energy. Overall, we are looking for such a coloring that satisfies all hard constraints within hypernodes and minimizes the weights of violated constraints among hypernodes so as to reduce the overall conflicts of scanning common regions among overlapping sectors.

We can model the scan scheduling problem as follows. 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,  $v_1, v_2, \dots, v_k$ , 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 sectors of sensors  $u$  and  $v$ . Every hypernode is given a total  $T$  available colors to use and a sensor activation ratio  $\alpha \leq 1$ .  $|T|$  corresponds to a schedule length.  $\alpha$  captures the frequency of how often a sensor should be active, and  $1 - \alpha$  is the frequency determining how often a sensor should turn itself off to save energy. Given the total available colors  $T$ , the weighted multi-coloring problem is to color the nodes following these criteria. First, exactly  $\lceil \alpha T \rceil$  colors are used within a hypernode. This means that a node may have more than one color if the available colors are more than the nodes in a 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 have any color in common. Third, minimizing conflicts between pairs of nodes in different hypernodes, so the total weight of constraint violations going 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  $T$  and such a schedule satisfying the criteria.

#### 4. Low Overhead Distributed Algorithms

The two algorithms we will discuss are synchronous [Tel, 2000], in that all distributed processes proceed in synchronized steps. In each step, a process

sends and receives (zero or more) messages and then performs local computations, i.e., changing local state, if necessary.

#### 4.1 Distributed breakout algorithm (DBA)

To understand DBA, we first discuss the original centralized breakout algorithm [Morris, 1993]. The algorithm is a local search method for CSP, equipped with an innovative scheme of escaping local minima. Given a CSP, the algorithm first assigns a weight of one to all constraints. It then selects a value for each 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 the chosen variable. The algorithm continues to select a variable and change its value 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 synchronized distributed breakout algorithm (DBA) [Yokoo, 2001; Yokoo and Hirayama, 1996]. 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 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 at 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 1.

The DBA algorithm was not designed for weighted constraint problems, in which constraints carry weights. We can extend DBA to weighted CSP as follows. The initial value of a constraint weight for the algorithm is the weight of the constraint in the original problem. Whenever the weight of a constraint needs to be increased in the algorithm, the constraint weight in the original problem is added. This way, the weight increase reflects the actual constraint weight. We use this extended version of DBA in our experiments.

#### 4.2 Distributed Stochastic Algorithm (DSA)

Distributed stochastic algorithm (DSA) is uniform [Tel, 2000], in that all processes are the same and have no identities to distinguish one another. It is

---

**Algorithm 1** 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

```

---

also synchronous in principle [Tel, 2000], 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, e.g., changing local state if necessary. Note that synchronization in DSA is not crucial since it can be achieved by a synchronization mechanism [Tel, 2000].

The idea of DSA and its variations is simple [Fabiunke, 1999; Fitzpatrick and Meertens, 2001]. 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 2.

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 1.1 lists five possible strategies for value change, leading to five different 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 ex-

---

**Algorithm 2** 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 1.1)
end while

```

---

cept 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 exists 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 or  $\Delta > 0$ . 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 increases 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. Among these five different variations, DSA-B (or CFP in [Fitzpatrick and Meertens, 2001]) is the best on regular graph coloring problems [Zhang et al., 2002]. In this research, we adopt DSA-B for our distributed scan scheduling, and simply refer to it as DSA in the rest of the discussion.

To change the degree of parallel executions, an agent may switch to a different DSA algorithm, or change the probability of parallelism  $p$  that controls the

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 1.1. Next value selection in DSAs. Here C stands for conflict,  $\Delta$  is the best possible conflict reduction between two steps,  $v$  the value giving  $\Delta$ , 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.

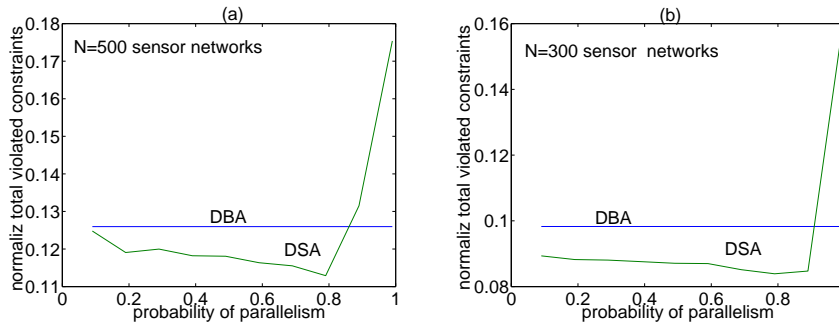


Figure 1.1. Phase-transition behavior of DSA on distributed scan scheduling,  $T = 6$ . The normalized total violated constraints is the ratio of the total weight of violated soft constraints the solutions found by DSA to the weight of all soft constraints.

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 [Tel, 2000] 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.

## 5. Threshold Behavior of DSA

In our previous study, a phase-transition behavior of DSA was observed on regular coloring problems [Zhang et al., 2002]. Here we consider this issue

again on distributed, weighted multi-coloring problems generated from our scan scheduling application. The objective is to better understand the behavior of DSA related to the degree of parallel executions among distributed agents.

A phase transition refers to the phenomenon of a system in which a global property changes rapidly and dramatically when an order parameter goes beyond a critical value. A simple example is the melting of ice with rising temperature. When the temperature reaches above the melting point, a phase transition from ice to water occurs. For our problem here, the system property is the average weight of violated soft constraints in DSA solutions normalized by the weight of all soft constraints. The control parameter is the degree of parallel execution  $p$ .

In our experiments, we set the sensing radius of a sensor to one unit, and used a square of  $10 \times 10$  units as the area to be monitored. The number of sensing sectors is 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 changed the probability of parallel executions  $p$  from 0.1 to 0.99, with an increment of 0.01. For each  $p$ , we generated 100 problem instances and converted them into multi-coloring problems. We evaluated the performance of DSA when it reached relatively stable states or states near equilibria. Specifically, we ran DBA to the point where its performance does not change significantly from one step to the next. On all network sizes we considered, DSA's performance seems to stabilize after 256 steps. Similar results have been observed after 1024, 2048 and longer steps. In the rest of this paper, we report the results at 256 steps.

In our first experiment, we used 500 sensors and six allowed colors, i.e.,  $T = 6$ . We set the sensor activation ratio  $\alpha = 2/3$  so that four colors will be used for a hypernode with three nodes in it. Following the hard constraints within a hypernode, i.e., a node must have at least one color and no two nodes can share a color, this means that one node must have two colors. Figure 1.1(a) shows the average performance of DSA, controlled by the probability of parallelism  $p$ . The performance is measured by the ratio of the total weight of violated soft constraints in the solutions found by DSA to the weight of all soft constraints. As the figure shows, the performance of DSA experiences a sharp and dramatic transition when the probability of parallelism  $p$  increases beyond 0.8. One interesting observation from this experiment, as well as other similar ones such as that in Figure 1.1(b), is that for a fixed sensor density, DSA reaches its best performance before the phase transition occurs.

Moreover, the phase-transition behavior occurs at a higher probability  $p$  when the number of sensors or graph density decreases. One example can be found by comparing the phase transition points in Figures 1.1(a) and 1.1(b), where the network for Figures 1.1(a) has 500 nodes while that used for Figures 1.1(b) has only 300 nodes, all of them are randomly placed in the detection

area. The phase-transition point in Figures 1.1(a) appears sooner than that in Figures 1.1(b). This result indicates that the more constrained the underlying graph is, the sooner the phase-transition behavior occurs. It seems to be an important, but difficult, problem to analytically determine the phase transition location for a given graph. Nevertheless, our experimental results provide some qualitative information on how the degree of parallel executions should be chosen. The tighter the constraints in a given problem or the higher the connectivity of a constraint graph, the smaller the degree of parallel executions should be. Similar phase-transition patterns were observed on the same sensor networks but with more allowed colors, for instance  $T = 18$ .

The phase transition result here is qualitatively similar to that in [Zhang et al., 2002] where regular graph coloring was considered. This implies that the phase-transition behavior of DSA persists in these two problems. However, although on both regular coloring and multi-coloring problems DSA's performance degenerates significantly when its degree of parallelism is too high, the algorithm pans out better on multi-coloring problems than on regular coloring problems. On regular graph coloring problems, DSA's degenerated performance may be much worse than that of a random coloring, while on multi-coloring problems the algorithm's degenerated performance is still better than that of a random coloring. For instance, on a 300-node multi-coloring problem, the average normalized solution quality of random multi-coloring is more than 0.65 while the averaged normalized quality of degenerated solution of DSA when  $p = 0.99$  is less than 0.16.

## 6. DSA vs. DBA on Solution Quality

We now directly compare DBA and DSA on distributed multi-coloring problems. We used the same set of parameters as used for studying phase transitions in Section 1.5. Specifically, each sensor has three equal sectors, and its sensing radius is set to one unit. Sensors are then randomly and uniformly placed in an area of  $10 \times 10$  square with arbitrary orientations. We experimented with different values of maximal allowed colors and different sensor activation ratio  $\alpha$ . In the rest of this section, we report the results using  $T = 6$  and  $T = 18$  with  $\alpha = 2/3$ .

### 6.1 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 multi-coloring problems produced from sensor networks of various sizes. We changed the density of multi-coloring graphs by

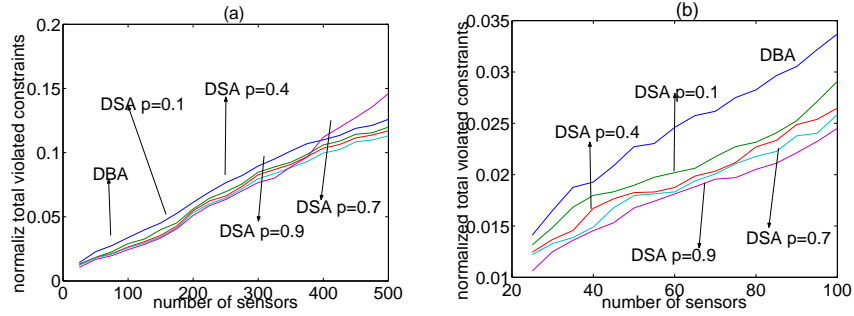


Figure 1.2. DSA vs. DBA in terms of number of sensors,  $T = 6$ .

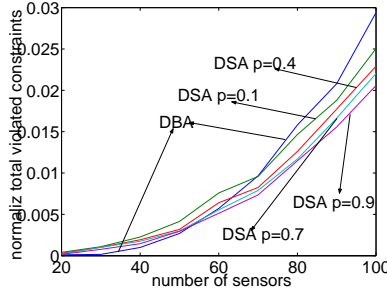


Figure 1.3. DSA vs. DBA in terms of number of sensors,  $T = 18$ .

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. Figure 1.2 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. Figure 1.2(a) shows the result in the whole range of 25 to 500 sensors and Figure 1.2(b) expands the results of Figure 1.2(a) in the range of 25 to 100 sensors,

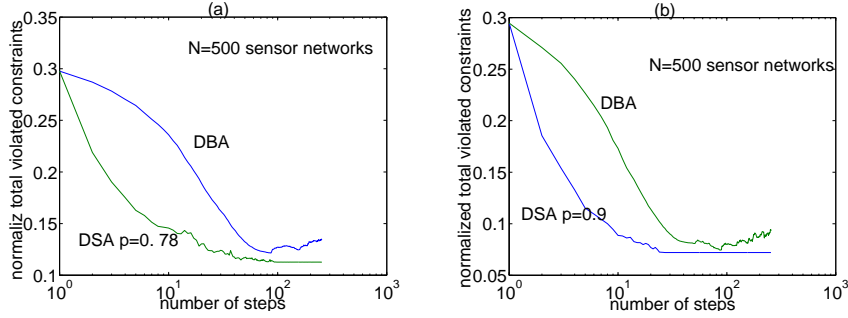


Figure 1.4. Anytime performance of DSA and DBA in dense sensor networks,  $T = 6$ .

As analyzed in [Zhang et al., 2002], DBA may perform better than DSA on underconstrained problems. An underconstrained scan scheduling problem may be created when more colors are available. Indeed, when we increase the number of allowed colors (targeting schedule cycle length) to eighteen ( $T = 18$ ), DBA outperforms DSA on sparse networks with less than 50 sensors. This result is shown in Figure 1.3, 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 parallel executions is not too high in the whole 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. For instance, DSA with  $p = 0.9$  becomes the worst of all when there are more than 400 sensors (Figure 1.2). This means that the denser the sensor networks are, the smaller the parallel degree  $p$  should be. 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 inherited 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. This result is in line with that in [Zhang et al., 2002].

## 6.2 Anytime performance

An important feature of our 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

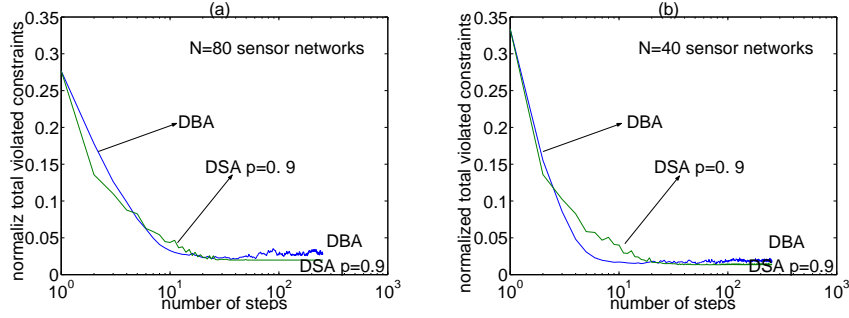


Figure 1.5. Anytime performance of DSA and DBA in sparse sensor networks,  $T = 6$ .

object tracking. Therefore, the algorithm for the distributed scan scheduling must have anytime property, i.e., the algorithm can be stopped at any time during its execution and is able to provide a feasible solution at that point. 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. 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  $10 \times 10$  grid, with results averaged over 100 trials.

We first considered dense networks with  $N = 500$  and  $N = 300$  sensors. Based on the phase-transition results in Section 1.5, 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. The experimental results are in Figure 1.4. As the results show, DSA performs much better than DBA in both anytime performance and final solution quality.

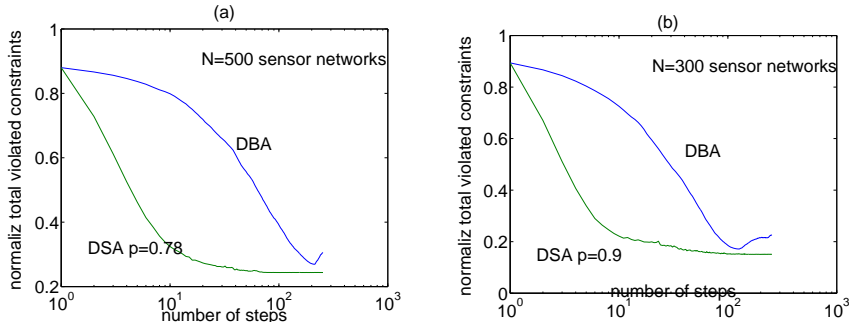


Figure 1.6. Anytime performance of DSA and DBA in dense sensor networks,  $T = 18$ .

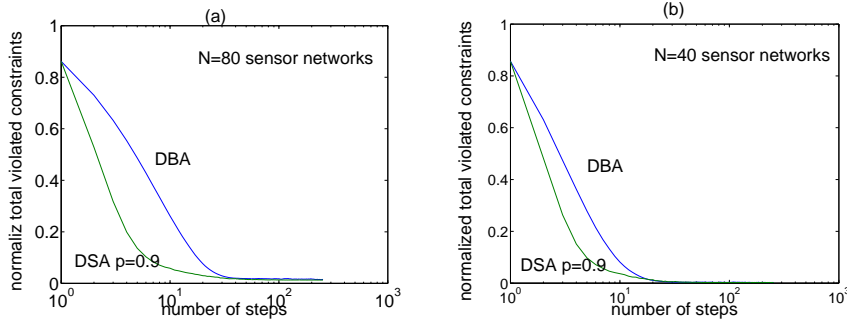


Figure 1.7. Anytime performance of DSA and DBA in sparse sensor networks,  $T = 18$ .

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 use  $p = 0.9$  for our two sparse networks. The results, averaged over 100 trials, are in Figure 1.5. 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 Figures 1.6 and 1.7, 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. DSA vs. DBA on Communication Cost

We have so far concentrated on the solution quality of DBA and DSA without paying any attention to their communication cost. As mentioned earlier, communication in a sensor network has an inherited 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 may not have to send a message in a step in DSA if it does not change its value. In an extreme case, an agent will not change its value if it is at an local minimum. If solution quality of DSA improves over time, its

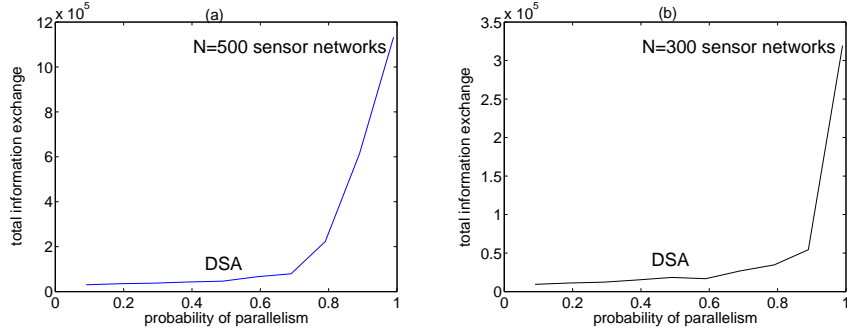


Figure 1.8. Communication-cost phase transitions of DSA on scan scheduling,  $T = 6$ .

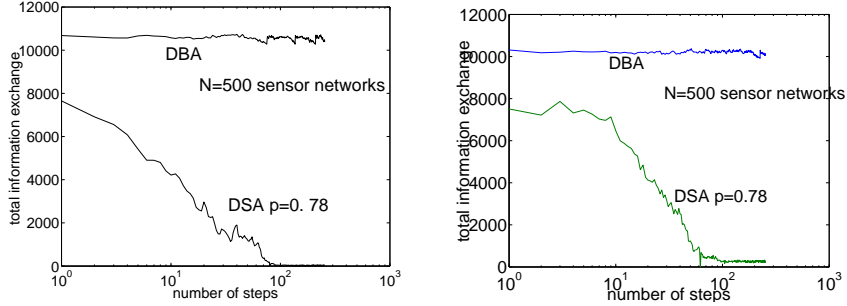


Figure 1.9. Communication cost of DSA and DBA,  $T = 6$  (left) and  $T = 18$  (right).

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 degrees of parallel executions of the agents. The higher the parallel probability  $p$  is, the higher the communication cost will be. As shown in [Zhang et al., 2002], 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. Figure 1.8 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. Figure 1.9 shows the results evaluating DBA and DSA with probability  $p = 0.78$

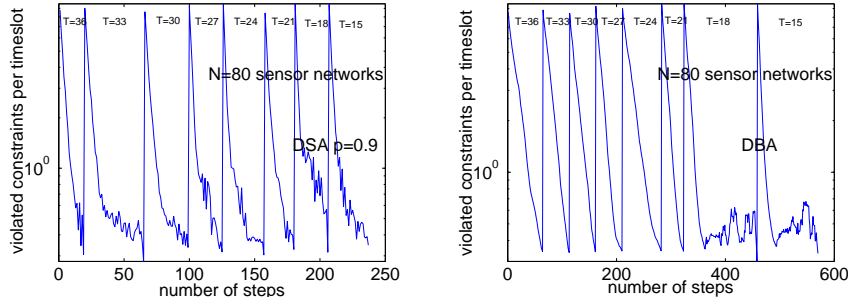


Figure 1.10. Finding best possible schedule using DSA (left) and DBA (right),  $N = 80$ .

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

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

## 8. Solving Scheduling Problem

Based on the results from Sections 1.5 to 1.7, we now apply DSA and DBA to dealing with two related problems at the same time, finding the shortest scan cycle length  $T$  and making 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 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.

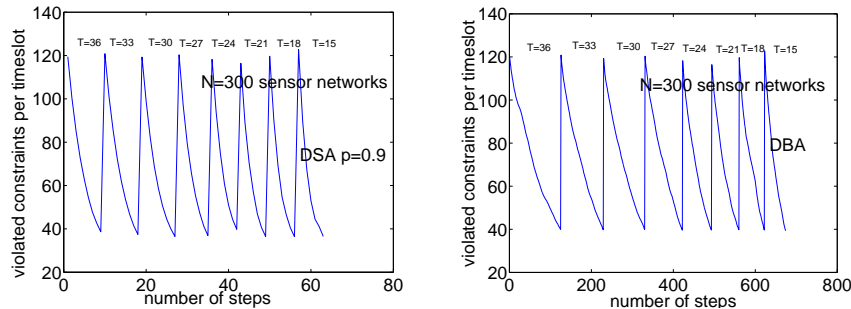


Figure 1.11. Finding best possible schedule using DSA (left) and DBA (right),  $N = 300$ .

Figures 1.10 and 1.11 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 an initial  $T = 36$ , and reduced the value of  $T$  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 (Figure 1.10), DSA finds a targeting schedule of length  $T = 15$  in 242 steps, while DBA needs 588 steps. On the  $N = 300$  network (Figure 1.11), 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, such as in wireless communication environments, DSA is definitely the algorithm of choice for the problem.

## 9. Conclusions

Motivated by real applications of multiagent systems in sensor networks, we studied low-overhead distributed algorithms for solving distributed constraint satisfaction and optimization problems. We analyzed and compared distributed stochastic algorithm (DSA) [Fabiunke, 1999; Fitzpatrick and Meertens, 2001; Zhang et al., 2002] and distributed breakout algorithm (DBA) [Morris, 1993; Yokoo, 2001; Yokoo and Hirayama, 1996; Zhang and Wittenburg, 2002] on a distributed scheduling problem in sensor networks. We specifically investigated the relationship among the degree of parallel executions, constrainedness of underlying problems, and DSAs' 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 exe-

utions increases beyond a critical point. On the other hand, we also 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 one considered in this paper, DSA is the algorithm of choice.

## Acknowledgment

This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

## References

- Fabiunke, M. (1999). Parallel distributed constraint satisfaction. In *Proc. Intern. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA-99)*, pages 1585–1591.
- Fitzpatrick, S. and Meertens, L. (2001). 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.
- Morris, P. (1993). The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 40–45, Washington, DC.
- Tel, G. (2000). *Introduction to Distributed Algorithms*. Cambridge University Press, 2 edition.
- Yokoo, M. (2001). *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer Verlag, Berlin, Heidelberg, New York.
- Yokoo, M. and Hirayama, K. (1996). Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96)*, pages 401–408.
- Zhang, W., Wang, G., and Wittenburg, L. (2002). Distributed stochastic search for distributed constraint satisfaction and optimization: Parallelism, phase transitions and performance. In *Proc. AAAI-02 Workshop on Probabilistic Approaches in Search*, pages 53–59.
- Zhang, W. and Wittenburg, L. (2002). Distributed breakout revisited. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 352–357.