*Proc. AAAI-02, to appear.*

# Distributed Breakout Revisited

**Weixiong Zhang** and **Lars Wittenburg**
Computer Science Department
Washington University
St. Louis, MO 63130
Email: {zhang,larsw}@cs.wustl.edu

## Abstract

Distributed breakout algorithm (DBA) is an efficient method for solving distributed constraint satisfaction problems (CSP). Inspired by its potential of being an efficient, low-overhead agent coordination method for problems in distributed sensor networks, we study DBA's properties in this paper. We specifically show that on an acyclic graph of $n$ nodes, DBA can find a solution in $O(n^2)$ synchronized distributed steps. This completeness result reveals DBA's superiority over conventional local search on acyclic graphs and implies its potential as a simple self-stabilization method for tree-structured distributed systems. We also show a worst case of DBA in a cyclic graph where it never terminates. To overcome this problem on cyclic graphs, we propose two stochastic variations to DBA. Our experimental analysis shows that stochastic DBAs are able to avoid DBA's worst-case scenarios and has similar performance as that of DBA.

## 1 Introduction and Overview

Our primary motivation of studying distributed breakout algorithm (DBA) [9; 11] is to apply it as a simple, low-overhead method for coordinating agents in distributed sensor networks [12]. One important class of problems among distributed agents is the coordination of their distributed actions in such a way that overall inter-agent constraints are not violated. Such a problem can be captured as a distributed constraint satisfaction problem (CSP) [9].

DBA is a remarkable extension of breakout algorithm for centralized CSP [6]. Centralized breakout algorithm is a local search method with an innovative method for escaping local minima. This is realized by introducing weights to constraints and dynamically increasing some of the weights so as to force agents to dynamically adjust their values. It has been shown experimentally that on certain constraint problems, it is more efficient than local search algorithms with multiple restarts and asynchronous weak-commitment search [9; 11].



Despite their unique features and early success, breakout and distributed breakout algorithms have not been studied thoroughly. For example, their completeness is not fully understood and their complexity remains unknown. It is also not clear what constraint graph structures will render worst cases for these algorithms. To our best knowledge, the work on these two algorithms is limited to the original publications on the subject, specifically [6; 9; 11].

Motivated by our real-world applications of distributed sensor networks in which DBA can apply [12] and inspired by its possible great potential in solving large distributed CSP, we study DBA in this paper. After a brief overview of breakout and distributed breakout algorithms (Section 2), we analyze the completeness and computational complexity of DBA (Section 3). We prove that on acyclic constraint graphs, DBA is complete, in the sense that it is guaranteed to find a solution if one exists. We also show that its complexity, the number of synchronized distributed steps, is $O(n^2)$ on an acyclic graph with $n$ nodes. These analytical results reveal the superiority of DBA over conventional centralized and distributed local search, which is not complete on acyclic graph. In addition, we identify the best and worst arrangements for variable identifiers on acyclic graphs, which are critical elements of the algorithm. These results indicate that DBA is an efficient, low-overhead method for self-stabilization [8] in tree-structured distributed systems. Furthermore, on cyclic graphs, we construct a case in which DBA is unable to terminate, leading to its incompleteness in this case. To avoid DBA's worst-case behavior on cyclic graphs, we introduce stochastic features to DBA (Section 4). We propose two stochastic variations to DBA and experimentally demonstrate that they are able to increase DBA's completeness on cyclic graphs and have similar anytime performance as the original algorithm.

Finally, we discuss previous related work in Section 5 and conclude in Section 6.

## 2 Breakout and Distributed Breakout

The breakout algorithm [6] 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

**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
```

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) [9; 11]. 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. For simplicity, we assume each step is synchronized among the agents. This assumption can be lifted by a synchronization mechanism [8].

In the description of [9; 11], 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 instances, 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 worth pointing out that the node, or agent, identifiers are not essential to the algorithm. They are only used to set up a priority between two competing agents for tie breaking. As long as such priorities exists, node identifiers are not needed.

## 3 Completeness and Complexity

In this section, we study the completeness and computational complexity of DBA on binary constraint problems in which no constraint involves more than two variables. This is not a restriction as a non-binary constraint problem can be converted to a binary one with cycles [1; 7]. One advantage of using binary problems is that we can focus on the main features of DBA rather than pay attention to the degree of constraints of the underlying problem. In the rest of the paper, we use constraint problems to refer to binary problems if not explicitly stated. In addition, 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 variables, nodes and agents interchangeably in our discussion.

### 3.1 Acyclic graphs

First notice that acyclic graphs are 2-colorable. Thus, any acyclic constraint problem must have a satisfying solution if the domain size of a variable is at least two. In addition, larger domains make a problem less constrained. Therefore, it is sufficient to consider acyclic constraint problems with variable domains no more than two.

To simplify our discussion and for pedagogical reasons, 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 1** *On a chain, DBA will not visit the same problem state more than once.*

*Proof*: Assume the opposite, i.e., DBA can visit a state twice in a process as follows, $S_x \rightarrow S_y \rightarrow \cdots \rightarrow S_z \rightarrow S_x$. 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 2** *On a chain of $n$ variables, each of which has a domain size at least two, DBA can increase a constraint weight to at most $\lfloor n/2 \rfloor$.*

*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 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.

**Theorem 1** *On a chain of $n$ nodes, 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.*

*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 domain sizes less than two. 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 $O(n)$ 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. 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 on a chain. If DBA does not find a solution in $n^2$ steps, it can terminate with an answer of no solution. 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 the 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 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 $n_1 \leq n_2$. 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 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.

**Theorem 2** *On an acyclic graph with $n$ nodes, 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.*

The above completeness result can be directly translated to centralized breakout algorithm, leading to its completeness on 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 on acyclic graphs, which are not complete even on a chain.

Our experimental results also show that the number of steps taken by DBA is much smaller than the $n^2$ upper bound, as shown in Figure 1. 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 identi-

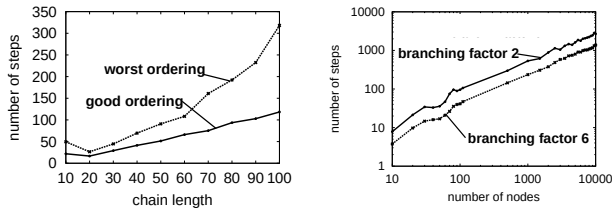


Figure 1: The number of steps taken by DBA on chains with the best and worst variable identifier arrangements (left) and on trees with worst identifier arrangements (right).

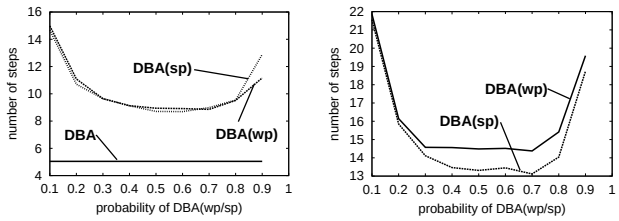


Figure 3: Steps taken by DBA and variants on the example of Figure 2 with random initial assignments (left) and the specific assignment of Figure 2 (right).

fier arrangements for chains (Figure 1 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 figure shows, 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 (Figure 1 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 acyclic graphs.

### 3.2 Cyclic graphs

Unfortunately, DBA is not complete on cyclic constraint graphs. This will include non-binary problems as they can be converted to binary problems with cycles. This is also the reason that breakout algorithm is not complete on Boolean satisfiability with three variables per clause [6], which is equivalent to a constraint with three variables.

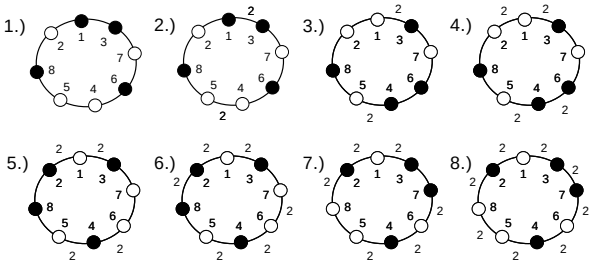


Figure 2: A worst case for DBA on a ring.

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 (black and white), as shown in Figure 2, where the node identifiers and constraint weights are respectively next to nodes and edges. Figure 2(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 Figure 2(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 counterclockwise in Figure 2(3). The rest of Figure 2 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.

## 4 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.

### 4.1 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 [3; 4; 13], 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 variable identifiers are needed, which may be important for some applications where node identifiers across 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.

### 4.2 DBA(wp) versus DBA(sp)

We first study the two variants on the example of coloring an 8-node ring of Figure 2. In the first set of tests, node identifiers and initial colors are randomly generated and 10,000 trials are tested. DBA is unable to terminate on 15% of the total trials after more than 100,000 steps[1], while on the other 85%

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

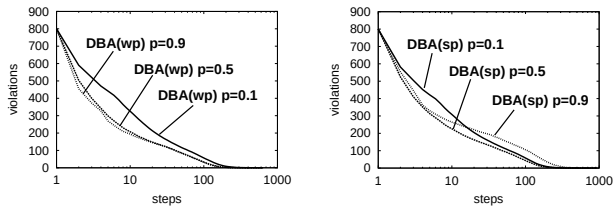


Figure 4: DBA(wp) and DBA(sp) on grid $20 \times 20$ and $k = 4$.

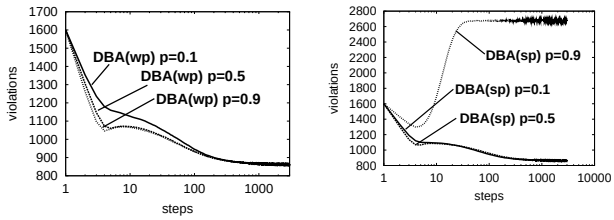


Figure 5: DBA(wp) (left) and DBA(sp) (right) on graph with 400 nodes and k=8.

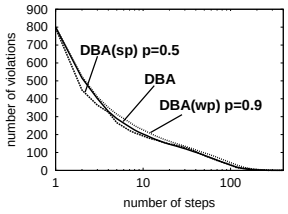


Figure 6: DBA and random DBAs on grid $20 \times 20$ and $k = 4$.

of the trials DBA finds a solution after 5 steps on average as shown in Figure 3(left). In contrast, DBA(wp) and DBA(sp) always find solutions but require almost twice as many steps on average with the best probability around $0.6$.

In the second set of tests, we use the exact worst-case initial assignment as shown in Figure 2. As expected, DBA failed to terminate. DBA(wp) and DBA(sp) find all solutions on 1,000 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 Figure 3(right).

Next we study these two families of variants on grids, graphs and trees. We consider coloring these structures using 2 colors. For grids, we consider $20 \times 20$, $40 \times 40$, and $60 \times 60$ grids with connectivities equal to $k = 4$ and $k = 8$. To simulate infinitely large grids in our experiments, we remove the grid boundaries by connecting the nodes on the top to those on the bottom as well as the nodes on the left to those on the right of the grids to create $k = 4$ grid. For $k = 8$ grid, we further link a node to four more neighbors, one each to the top left, top right, bottom left and bottom right. This renders the problem overconstrained for two-coloring. Hence, the algorithms may only try to improve the solution quality by minimizing the number of violated constraints.

The results of $20 \times 20$ grids with $k = 4$ are shown in Figure 4, averaged over 2,000 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 generate 2,000 graphs with 400 nodes with an average connectivity per node equal to $k = 4$ and $k = 8$ by adding, respectively, 1,600 and 3,200 edges to randomly picked pairs of unconnected nodes. These two graphs are generated to make a correspondence to the grid structures of $k = 4$ and $k = 8$ considered previously, except that both random graphs are not two-colorable. All algorithms are applied to the same set of graphs for a meaningful comparison. Figure 5 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 is close to distributed stochastic algorithm [3; 4; 13], the results here are in line with those of [13].

We also consider 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, they all find 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.

### 4.3 DBA(wp) and DBA(sp) versus DBA

The remaining issue is how DBA(wp) and DBA(sp) compare with DBA. Here we use the best parameters for these two variants from the previous tests and compare them directly with DBA. We average the results over the same sets of problem instances we used in Section 4.2. Figures 6, 7 and 8 show the experimental results on grids, random graphs and trees, respectively. With their best parameters, DBA(wp) and DBA(sp) appear to be compatible with 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.

## 5 Related Work and Discussions

It is well known that acyclic constraint problems can be solved in linear time by an arc consistency algorithm followed by a backtrack-free value assignment [5]. However, on acyclic graphs there exists no uniform distributed algorithm that is self-stabilizable in the sense that it is guaranteed to reach a solution from an arbitrary initial state [2]. In a uniform distributed algorithm [8], all nodes execute the same procedure and two nodes do not differentiate themselves. Therefore, DBA is not a uniform algorithm as two adjacent nodes can differ from each other by their different priorities. It has also been shown that by introducing only one unique node to a constraint graph, an exponential-complexity

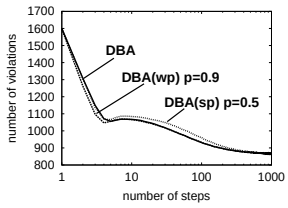


Figure 7: DBA and random DBAs on graph with 400 nodes and k=8.

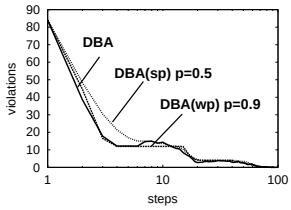


Figure 8: DBA and random DBAs on tree with depth $d = 4$ and branching factor $k = 4$.

self-stabilization algorithm exists [2]. In that regard, our results of DBA on acyclic graph indicate that node priorities are merely a tool for reducing complexity.

One related algorithm for distributed CSP is asynchronous weak-commitment (AWC) search algorithm [9; 10]. One major difference between AWC and DBA is that node priorities in AWC may change dynamically while node priorities in DBA are stationary. In contrast, constraint weights may increase dynamically in DBA while they are static in AWC. With a sufficient amount of memory to record the states (agent views) that an agent has visited, AWC is guaranteed to converge to a solution if it exists. In the worst case, the amount of memory required is exponential of the problem size. In contrast, DBA converges to a solution on acyclic graphs without any additional memory. It may be also the case that AWC does not require an exponential amount of memory to reach a solution on acyclic graph, an interesting future research topic.

Another related algorithm for distributed CSP is distributed stochastic algorithm (DSA) [3; 4; 13]. DSA is a family of stochastic search algorithms with two members being different from each other on the degrees of parallelism of agent actions. Contrasting to DBA, DSA does not require priorities among agents and is not guaranteed to find a solution either, even if one exists. Our limited experimental results showed that DBA is more efficient than DSA on acyclic graphs, which is supported by our analytical results in this paper, and on many underconstrained cyclic constraint problems. Our complete results comparing these two algorithms will be included in our final report of this research.

## 6 Conclusions

We closely examined the completeness and computational complexity of distributed breakout algorithm (DBA) in this paper. We showed that DBA is complete and has low polynomial complexity on acyclic graphs. This result is important as it shows the superiority of DBA over conventional local search, which does not guarantee the completeness even on a chain. The result also implies that DBA can be used as a method for self stabilization in tree-structured distributed systems. We also identified a simple worst-case node identifier arrangement on a ring in which DBA may not terminate. This helps to understand the behavior of DBA on graphs and non-binary constraint problems. We further proposed and experimentally demonstrated that randomization can overcome such worst-case situations without a significant penalty to DBA's anytime performance.

## Acknowledgment

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

## References

[1] F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proc. AAAI-98*, pages 310–318, 1998.

[2] Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 3(4), 2000.

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

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

[5] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[6] P. Morris. The breakout method for escaping from local minima. In *Proc. AAAI-93*, pages 40–45.

[7] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proc. ECAI-90*, pages 550–556, 1990.

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

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

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

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

[12] W. Zhang, Z. Deng, G. Wang, L. Wittenburg, and Z. Xing. Distributed problem solving in sensor networks. In *Proc. AAMAS-02*.

[13] W. Zhang, G. Wang, and 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*, to appear.