# Solving DisCSPs with Penalty Driven Search

**Muhammed Basharu, Ines Arana,** and **Hatem Ahriz**
School of Computing,
The Robert Gordon University, Aberdeen, United Kingdom
{mb, ia, ha}@comp.rgu.ac.uk

**Abstract**

We introduce the Distributed, Penalty-driven Local search algorithm (DisPeL) for solving Distributed Constraint Satisfaction Problems. DisPeL is a novel distributed iterative improvement algorithm which escapes local optima by the use of both temporary and incremental penalties and a tabu-like no-good store. We justify the use of these features and provide empirical results which demonstrate the competitiveness of the algorithm.

## Introduction

In distributed environments, many of the problems resolved by collaborative agents can be formalized as Distributed Constraint Satisfaction Problems (DisCSPs). In this formalisation, problems such as scheduling and resource allocation are decomposed into variables and constraints which are distributed amongst the agents involved. The problems are solved by a collaborative process in which agents try to reach agreements (i.e. satisfy constraints) by exchanging value assignments for their respective variables.

Modelling problems as DisCSPs facilitates the problem solving process by allowing agents to use the constraints in the problem to rule out non-solutions via constraint propagation. In many cases, however, problems cannot be solved by just constraint propagation, and seeking a solution by search is inevitable. A complete search for a solution poses many well known problems and, in distributed environments, these are magnified since the problem solving process involves the exchange of information between physically dispersed agents and reasoning with incomplete information. In this work, we propose a novel distributed search algorithm that aims to improve search efficiency by helping agents identify and avoid unprofitable areas of the search space and hence focus on regions where solutions may exist. The algorithm, Distributed Penalty-driven Local search (DisPeL) is a distributed iterative improvement algorithm which escapes local optima by the use of both temporary and incremental penalties and a tabu-like no-good store. We discuss how these features influence the search process and show their efficiencies with experiments on randomly generated problems.

The rest of this paper is structured as follows. We start with an overview of distributed constraint satisfaction. Then we discuss the strategy employed in DisPeL and provide some experimental justification for some of its components. Next, we present results of empirical evaluation on distributed graph coloring problems and random non-binary DisCSPs. And finally, we conclude the paper with a brief summary.

## Distributed Constraint Satisfaction

A Constraint Satisfaction Problem (CSP) is formally defined as a triple (V, D, C) comprising a set of variables (V), a set of domains (D) listing possible values that may be assigned to each variable, and a set of constraints (C) on values that may be simultaneously assigned to the variables. The solution to a CSP is a complete assignment of values to variables satisfying all the constraints. In a DisCSP, variables and constraints are already distributed among a network of homogeneous agents. Each variable belongs to only one agent, while each agent may have more than one variable[1]. DisCSPs are solved by a collaborative process, where each agent strives to find assignments for its variables that satisfy all attached constraints.

The key distinguishing features of the DisCSP are privacy and partial knowledge of a problem (Yokoo *et al.* 1992). Agents are only aware of the domains and the constraints for the variables they represent. Privacy requirements also mean that an agent can not know about the domain of another agent or about the constraints any other agent is involved in. In addition, with privacy, the only information an agent is permitted to reveal to other agents is the current assignments to its variables. These features distinguish distributed constraint satisfaction from parallel or distributed approaches for solving traditional CSPs.

The DisCSP was formally described in the seminal work by Yokoo et al.(1992), and the field was influenced early on by search algorithms introduced by those authors. The portfolio of search algorithms has grown considerably in recent times but the key challenge, and what differentiates the algorithms, remains how dead-ends or local op-

[1]In this paper we assume that each agent holds just one variable. Therefore, we use the terms variable and agent interchangeably to refer to the variables that agents represent.

tima[2] are dealt with. In distributed backtracking algorithms such as Asynchronous Weak Commitment Search (Yokoo & Hirayama 2000) and Dynamic Distributed Backtrack (Bessiere, Maestre, & Meseguer 2001), new constraints are generated out of the deadlock situations. These help identify sources of deadlocks and help the algorithms avoid revisiting deadlock states.

In distributed iterative improvement algorithms, several alternative mechanisms have been introduced that aim to help identify sources of deadlocks and initiate the right actions for their resolution. For example, in the Distributed Breakout Algorithm (DBA) (Hirayama & Yokoo 2005) agents continually increase the importance of the constraints associated with the deadlocks and therefore are able to focus attention on their resolution. Other algorithms in the same class, such as the Distributed Stochastic Search (Zhang *et al.* 2005), rely on stochastic decisions to avoid and escape from local optima. This was shown to be an efficient strategy for distributed constraint optimisation problems; converging quicker than DBA on a restricted class of problems where the aim is to satisfy as many constraints as quickly as possible rather than solve the whole problem.

## Distributed Penalty Driven Local Search

### Strategy for Resolving Local Optima

DisPeL is a greedy algorithm that starts off with an initial random solution which agents try to improve in successive iterations by selecting values that minimize the number of constraints violated. When agents are stuck in deadlocks and unable to improve the solution, DisPeL uses either of two penalties - temporary and incremental - to resolve the conflicts creating the deadlocks. The temporary penalty is first used to perturb deadlocked agents. While the incremental penalty is used to avoid assignments that are associated with deadlocks if perturbations did not work.

The temporary penalty is imposed on the current value assigned to a deadlocked variable, and it is disposed off immediately after it is used. The purpose of the temporary penalty is to create a perturbation, by forcing agents to choose values other than their current assignments, which pushes agents out of the current point in the search space and encourages exploration of distant locations. Empirical evidence (discussed later) shows that when the temporary penalty is used, agents are able to resolve some conflicts immediately and, critically, do not often create new constraint violations in other parts of the problem. The temporary penalty has a fixed size $(t > 1)$ and its value may be problem dependent. In our work so far, we found that in some problems, such as distributed graph coloring, a small temporary penalty ($t = 3$) worked best. In other problems, like the car sequencing problem(Parrello, Kabat, & Wos 1986), a large temporary penalty is more suitable. This may be related to the neighborhood structures or to the nature of the constraints involved.

An incremental penalty is attached to each value in a variables domain, and it serves as search memory for agents. When a deadlock is not resolved by perturbation, incremental penalties attached to the values are steadily increased; contorting the shape of the plateaux occupied, thereby allowing agents to resume the search. The incremental penalties are reset to zero (i) periodically during the search and, (ii) individually by agents whenever they find consistent assignments for their variables. In the latter case, penalties are reset because they are assumed to have become redundant. Whereas in the former case, penalties are reset periodically to limit their impact on the objective landscape and leave paths to potential solutions unhindered.

The temporary and incremental penalties are included in each agent's objective function, which is defined as follows:

$$h(d_i) = v(d_i) + p(d_i) + \begin{cases} t & \text{with temporary penalty} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where:
$v\,(d_i)$ is the number of constraints violated with domain value $d_i$,
$p(d_i)$ is the incremental penalty associated with $d_i$,
$t$ is the temporary penalty imposed on $d_i$ and $t > 1$.

A no-good store is used primarily to help agents decide what penalties to apply when deadlocks are encountered. Each agent maintains a no-good store to keep track of recent conflict states, storing a maximum of *N* states at any point in time, where *N* is the number of neighbors the agent has. This value is used to retain structural aspects of the individual problem into the algorithm, and to avoid the need for parameter tuning as the problem sizes change. The nogoods here are records of an agent's conflict state (i.e. the AgentView comprising all its neighbors' assignments) and are not taken as new constraints. Therefore, no additional links are created between previously unrelated agents. Nogoods are also not used to prevent agents from revisiting previous states as in Tabu search (Glover 1990).

### Agent Behavior

Agents execute processes outlined in Figures 1, 2, and 3. At initialisation, agents first create a static ordering using the Distributed Agent Ordering scheme (Hamadi 2002). This induces an ordering among agents that allows unconnected agents to act concurrently. Each agent uses the scheme locally to determine its set of higher and lower priority neighbors using their lexicographic IDs.

After initialisation, agents take turns acting; each agent waits to receive the values selected by all its higher priority neighbors. Upon this, an agent typically selects a value in its domain that minimizes equation (1) and sends this value to all its neighbors (Figure 2, lines 3-6).

Conflict resolution is initiated whenever an agent finds itself at a quasi-local-optimum. Here, a quasi-local-minimum is defined as a situation where the AgentView of an agent

[2] A Pareto optimal situation with some unresolved constraints.

```
 1  initialise
 2  do
 3    when active
 4      rpCounter++
 5      if rpCounter = resetPeriod
 6        reset incremental penalties
 7        rpCounter = 0
 8      evaluate state
 9      if penalty request received
10        respond_to_message()
11      else
12        if current value is consistent
13          reset incremental penalties
14          send current value to neighbors
15        else
16          resolve_conflict()
17        end if
18      end if
19    return to inactive state
20  until terminate
```

Figure 1: DisPeL: Main agent loop

```
 1  procedure resolve_conflict()
 2    if AgentView(t) ≠ AgentView(t-1)
 3      select value minimizing eqn(1)
 4      send message(id, value, null)
 5      return
 6    end if
 7    if AgentView(t) is not in no-good store
 8      add AgentView(t) to no-good store
 9      impose temporary penalty on current value
10      select value minimizing eqn(1)
11      send message(id, value, addTempPenalty)
12    else
13      increase incremental penalty on current value
14      select value minimizing eqn(1)
15      send message(id, value, increasePenalty)
16    end if
17  end procedure
```

Figure 2: DisPeL: Initiating the conflict resolution process.

```
1  procedure respond_to_message()
2    if message is increase incremental penalty
3      increase incremental penalty on current value
4    else
5      impose temporary penalty on current value
6    end if
7    select value minimizing eqn(1)
8    send message(id, value, null)
9  end procedure
```

Figure 3: DisPeL: Responding to a penalty message received from a higher priority agent

with an inconsistent variable is unchanged in two consecutive iterations. To start conflict resolution, an agent first checks its no-good store to find out if the conflict has recently been encountered (Figure 2, line 7). If the conflict state is not in the no-good store, the agent imposes a temporary penalty on its current value and selects the value in its domain minimizing its objective function. Following that, the agent places the current no-good in the store and sends a message to all lower priority agents violating constraints with it requesting them to impose temporary penalties on their current values. At the same time, the agent sends its new value to all its neighbors. This first resolution phase is outlined in Figure 2 (lines 8-11).

If the conflict has been previously encountered, the agent moves into the second resolution phase outlined in Figure 2 (lines 13-15). The incremental penalty attached to the current domain value is increased by 1 and the agent selects the value minimizing its objective function. This new value is sent to all its neighbors, as well as a request for all lower priority neighbors to increase the incremental penalties attached to their current assignments.

When an agent receives a penalty request from a higher priority neighbor, it proceeds to impose the required penalty on its current value (Figure 1, line 10 and Figure 3). If an agent simultaneously receives messages to impose temporary penalties and increase incremental penalties, the temporary penalty request is ignored in favor of the increase.

## Justification for the Temporary Penalty

In earlier investigations for this work, we first looked at the immediate impact of different strategies and examined how effective they were in getting an algorithm out of a local optimum. Results from that study suggest that with a temporary penalty, 57% of constraints violated at the local optimum are resolved almost immediately. While its use caused new constraint violations in other parts of the problem 43% of the time. In contrast, the incremental penalty resolved 65% of constraints violated at the local optimum, causing new constraint violations 91% of the time.

We created two algorithms using each of these heuristics, i.e. one using the temporary penalty alone and the other using the incremental penalty alone, and compared their performance on random graph coloring problems. The results are shown in Table 1, and include a comparison with DisPeL. The average costs (i.e. the number of iterations) are from attempts on a set of 100 random graphs with a maximum of 5,000 iterations for each attempt.

| Heuristic | % solved | average cost |
|---|---|---|
| Temporary penalty alone | 76 | 729 |
| Incremental penalty alone | 96 | 255 |
| DisPeL | 99 | 180 |

Table 1: Performance of heuristics on random graph coloring problems (*n*=100, *k*=3, *degree*= 4.6).

The results show that with incremental penalties alone, one is able to solve nearly as many problems as with the final algorithm (i.e. DisPeL). While the temporary penalty, on its own, is considerably worse both in terms of effectiveness and efficiency. But, their combination creates an interesting synergy within DisPeL that improves search efficiency by over 40%. The contributions to the efficiency gains by the temporary penalty (and the accompanying no-good store) include the induced exploration and the reduced probability of oscillation by not causing new constraint violations while resolving existing conflicts.

## Determining the Optimal Reset Period

We choose to reset incremental penalties periodically in DisPeL, while aware of the potential risks of losing search experience in the process. However, there are arguments in favor of doing this in the literature. For example, in Morris' work on the centralized Breakout Algorithm (Morris 1995), it was argued that weights[3] can cause incompleteness as the changes in the objective landscape conspire to block paths to solutions. While Voudouris (1997) argues that penalties in the Guided Local Search algorithm do become redundant at some point and contends that periodically resetting penalties allows the search to revisit solutions penalised earlier, subsequently leading to an intensification of the search in areas around those solutions.

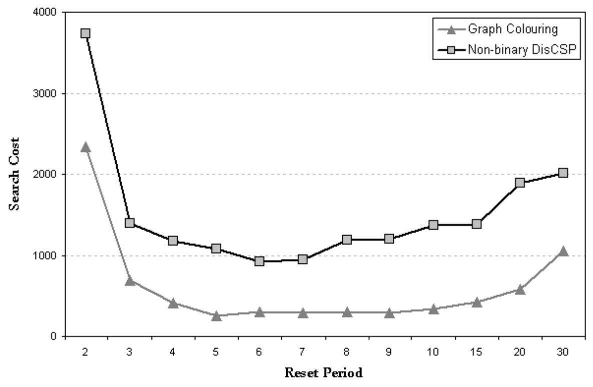

Figure 4: Average search costs with DisPeL on graph coloring problems $(n = 100, k = 3, degree = 4.7)$ and random 3-ary DisCSPs (75 variables, 150 constraints, 0.55 tightness, and 8 values in each variables domain) as the reset period changes. Each point is an average of 200 instances.

In Figure 4 we show results of experiments used to determine the optimal reset period for DisPeL, where we tested the algorithm on random graph coloring problems, as well as on random ternary DisCSPs, while varying the reset period. The plots show the average search cost of solving problems. As expected, the results indicate that when incremental penalties are reset frequently (i.e. reset period is less than 4) the algorithm solved fewer problems (not shown) and it had higher search costs on those solved. However, it appears that search costs are at a minimum when the reset period is 5 (for the graph coloring problems) or 6 (for the tenary problems) after which costs steadily increase. With high reset periods, the incremental penalties inhibit the algorithm's performance by obscuring the objective landscape to the extent that agents are constantly being pushed towards areas with higher constraint violations. The valleys and peaks of the landscape are gradually smoothed as more penalties are retained, therefore increasing the probability of the algorithm falling into a trap that causes it to roam about infinitely between non-solution states. Penalties start to dominate the objective function as they are retained, and as a result emphasis gradually shifts from minimising constraint violations to minimising the penalties. Incremental penalties are reset every six iterations in DisPeL irrespective of the the problem type and size (*resetPeriod* = 6 in Figure 1, line 5). This value is used for all the experimental evaluations reported in this paper.

[3]Weights are attached to constraints in the Breakout algorithm, with the goal of modifying the objective landscape for escaping local optima.

## Soundness and Completeness

DisPeL is sound because it does not return invalid solutions. This is so because it only reaches stable states when there are no constraint violations. Agents will retain their assignments (Figure 1, lines 12 - 14) if no constraints are violated, thus stabilizing the algorithm. And as long as a solution has not been found, agents will continue to perturb the state. Therefore, soundness is assured.

If a solution does not exist, the agents are unable to detect this and therefore the algorithm will run indefinitely. No-goods are not used as new constraints and besides, only a few are stored at any one time. As such, there is no opportunity to completely rule out infeasible areas of the search space. In addition, because incremental penalties are reset periodically and when agents find consistent assignments for their variables, they cannot provide any clues to insolubility. Therefore, the algorithm is incomplete.

## Empirical Evaluation

We have evaluated DisPeL on a number of problem types and present the results on experiments with distributed graph coloring and random non-binary DisCSPs. In this study, we were interested in its ability to solve problems and its efficiency in terms of the number of iterations typically required to solve the problems. Iteration count is used for evaluation here as the search cost because, as pointed out in (Ahuja & Orlin 1996), it is an independent performance metric which abstracts out the influences of implementations or the run time environment. Using the iteration count also allows for future comparison with our work, while serving as an approximation of the real costs in solving DisCSPs i.e. message count, for these algorithms.

We compared the algorithm's performance with that of the Distributed Breakout Algorithm (DBA) (Hirayama & Yokoo 2005) (the Single-DB version) on the same data. DBA is chosen for comparison because it has been established as one of the benchmarks with which distributed algorithms are compared. In addition, DBA and DisPeL are both incomplete, iterative improvement algorithms. To verify our implementation of DBA, we tested it with similar problems as those used in (Yokoo & Hirayama 1996) and achieved similar results.

## Distributed Graph Coloring Problems

An algorithm from (Fitzpatrick & Meertens 2002) was used to generate solvable 100-node problem instances for our experiments. The difficulty of instances is controlled using the average number of connections to each node (i.e. the degree) to study the behavior of both algorithms around the phase transition region (Cheeseman, Kanefsky, & Taylor 1991).

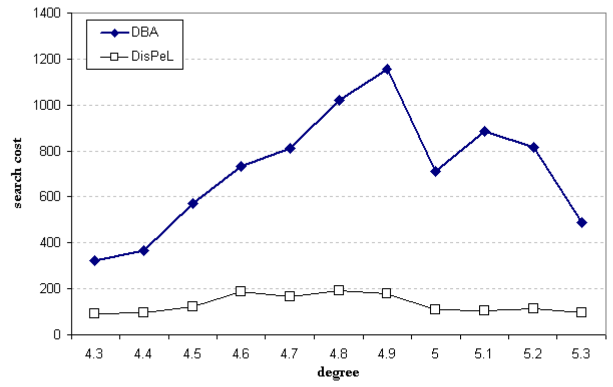

Figure 5: Behavior of DisPeL and DBA on random graphs (100 nodes and $k = 3$) as a function of connectivity.

Space limitations here prevent us from including details of the percentage of problems solved by both algorithms, but we include a plot of the median search costs of both algorithms in Figure 5. Both algorithms were each tested on 100 instances for each point in the plots, and attempts were terminated as failures after a maximum of 20,000 iterations for DBA and 10,000 iterations for DisPeL[4]. Both algorithms were able to solve nearly all problems in the test set, consistently above 95%, with little difference in performance on that criterion. But as evident in Figure 5, the median costs of finding solutions is significantly lower for DisPeL than for DBA. The performance gap is more pronounced in the phase transition region (i.e. between 4.6. and 4.9, see (Hogg 1996)) where DBA's median cost peaks at 1200 iterations, while the median cost for DisPeL in the same region is around 200 iterations.

## Random Non-Binary DisCSPs

Random non-binary problems where generated using a modified Model B (Palmer 1985), each defined by the tuple <n, d, a, c, t>, where *n* is the number of variables in the problem, *d* is the domain size, and *a* is the arity of all the constraints in the problem. In addition, *c* specifies the number of constraints in the problem and the tightness of each constraint is *t*. To ensure solubility, we included support values for each variable while generating the constraints.

Here, we report results from experiments on ternary (3-ary) problems in which we study the behavior of the algorithms as *n* increases. There are *2n* constraints in each problem and the constraint tightness is held constant at 0.55. In Figures 6 and 7, we plot the success rate and average cost respectively, on 100 problems for each point. We limited DisPeL to *100n* iterations and DBA to *200n* iterations with each attempt.

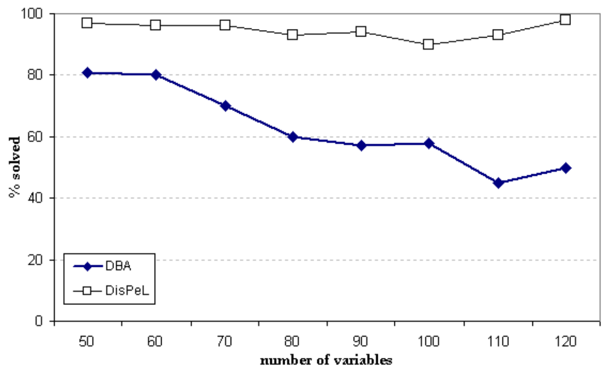

Figure 6: Fraction of random 3-ary DisCSPs ($< n, 8, 3, 2n, 0.55 >$) solved as problem size increases.

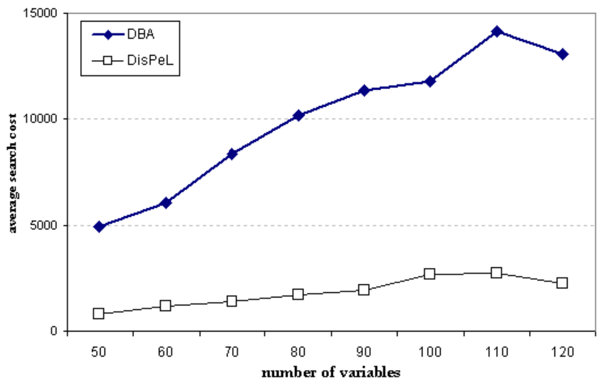

Figure 7: Average search costs required to solve the DisCSPs in Figure 6.

The results plotted in Figures 6 and 7 are consistent with those reported on the experiments on graph coloring problems in the previous section. They also show that as the problem size increases, DisPeL consistently outperforms DBA both in terms of the percentage of problems solved and average cost. In Figure 8 Run Length Distributions (Hoos & Stutzle 1999) for both algorithms on a single problem instance with constraints of mixed arities are plotted. These show the probability of solving problems with a given number of iterations from 500 attempts by each algorithm, with a maximum cut-off of 10,000 iterations for DisPeL and 20,000 iterations for DBA. Figure 8 suggests that DisPeL has a higher probability of finding solutions with fewer iter-

[4]DBA's two cycles (i.e. *wait_ok* and *improve?*) are equivalent to a single iteration in DisPeL.

ations than DBA. Further evidence is at the top end of the curves, where at the cut-off, DBA only found solutions to the problem in about 90% of its attempts compared to 99% for DisPeL.

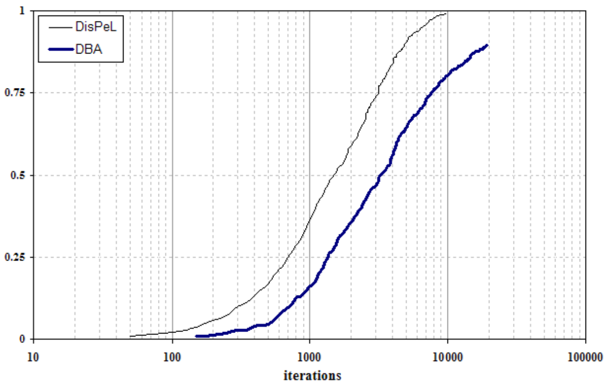

Figure 8: Run Length Distribution of DisPeL and DBA on a random non-binary CSP ($n$ = 80) with constraints of mixed arities (40 binary constraints, 100 3-ary constraints and 60 4-ary constraints) all with 50% constraint tightness.

## Summary

We have presented DisPeL, a novel algorithm for solving DisCSPs which uses penalties and a tabu-like no-good store in order to escape local optima. Two types of penalties (temporary and incremental) enhance search efficiency. The temporary penalty boosts search efficiency by helping agents to resolve some conflicts immediately and by minimizing the probability of conflicts being transferred to other parts of the problem while agents attempt to resolve those on hand. While, incremental penalties serve as a search memory for agents. We have justified their periodic disposal and explained how long-term retention of memory obscures the search landscape thus hindering overall performance. A limited form of learning is implemented with the use of a tabu-like no-good store which helps agents decide on what conflict resolution strategy to employ when stuck at local optima.

We are currently extending DisPeL for problems where each agent may have multiple local variables. In addition, we are also complementing our extensive evaluation to include other problem classes.

Results of empirical evaluations, as well as comparisons with DBA indicate that the strategy adopted in DisPeL is highly successful in solving random graph coloring and non-binary DisCSPs. DisPeL typically requires fewer iterations to solve the problems and it consistently finds more solutions. We have obtained similar results from our experiments with other problems such as the car sequencing problem and Schur's lemma.

## References

Ahuja, R. K., and Orlin, J. B. 1996. Use of representative operation counts in computational testing of algorithms. *INFORMS Journal of Computing* 8(3):318–330.

Bessiere, C.; Maestre, A.; and Meseguer, P. 2001. Distributed dynamic backtracking. In Walsh, T., ed., *Lecture Notes in Computer Science 2239 Springer (CP 2001)*.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on ArtificialIntelligence, IJCAI-91, Sidney, Australia*, 331–337.

Fitzpatrick, S., and Meertens, L. 2002. Experiments on dense graphs with a stochastic, peer-to-peer colorer. In Gomes, C., and Walsh, T., eds., *Proceedings of the Eighteenth National Conference on Artificial Intelligence AAAI-02*, 24–28. AAAI.

Glover, F. 1990. Tabu search - Part II. *ORSA Journal on Computing* 2(1):4–32.

Hamadi, Y. 2002. Interleaved backtracking in distributed constraint networks. *International Journal on Artificial Intelligence Tools* 11 (2):167–188.

Hirayama, K., and Yokoo, M. 2005. The distributed breakout algorithms. *Artificial Intelligence* 161(1–2):89–115.

Hogg, T. 1996. Refining the phase transition in combinatorial search. *Artificial Intelligence* 81:127–154.

Hoos, H. H., and Stutzle, T. 1999. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence* 112:213–232.

Morris, P. 1995. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 40–45.

Palmer, E. M. 1985. *Graphical evolution: an introduction to the theory of random graphs*. John Wiley & Sons, Inc.

Parrello, B. D.; Kabat, W. C.; and Wos, L. 1986. Job-shop scheduling using automated reasoning: a case study of the car-sequencing problem. *Journal of Automated Reasoning* 2(1):1–42.

Voudouris, C. 1997. *Guided local search for combinatorial optimisation problems*. Ph.D. Dissertation, University of Essex, Colchester, UK.

Yokoo, M., and Hirayama, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multi-Agent Systems*, 401–408. MIT Press.

Yokoo, M., and Hirayama, K. 2000. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* 3:189–212.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *12th International Conference on Distributed Computing Systems (ICDCS-92)*, 614–621.

Zhang, W.; Wang, G.; Xing, Z.; and Wittenburg, L. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161(1–2):55–87.