Distributed Guided Local Search for Solving Binary DisCSPs

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 Guided Local Search (Dist-GLS) algorithm for solving Distributed Constraint Satisfaction Problems. Our algorithm is based on the centralised Guided Local Search algorithm, which is extended with additional heuristics in order to enhance its efficiency in distributed scenarios. We discuss the strategies we use for dealing with local optima in the search for solutions and compare performance of Dist-GLS with that of Distributed Breakout (DBA). In addition, we provide the results of our experiments with distributed versions of random binary constraint satisfaction and graph colouring problems.

Introduction

In Distributed Constraint Satisfaction, a problem is partitioned amongst several physically dispersed agents, each responsible for finding a solution for its assigned sub-problem (Yokoo & Hirayama 2000). The key challenge in solving a Distributed Constraint Satisfaction Problem (DisCSP) is the restriction on what information may be revealed by agents to each other for reasons such as privacy or security; this limits the scope of inference available to agents for resolving constraint violations with other agents which are typically associated with local optima in the search for a solution.

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 constraints. In a DisCSP, each variable is assigned to only one agent, while agents may hold one or more variables. In this work, we focus on the case that all agents have one variable each. DisCSPs are solved by a collaborative process, where each agent strives to find assignments for its variables that satisfy all constraints.

Yokoo and Hirayama (2000) stress issues such as privacy and partial knowledge of the problem by agents in their justification for distributed constraint satisfaction. Agents can only reveal the values currently assigned to their variables during the search for solution, and nothing more. Agents are unaware of other agents' domains and constraints and make their decisions solely based on what is revealed to them by other agents. These restrictions become particularly important when the collaborative search process hits a local optimum. Each agent would already have selected values for its variables that minimise the number of constraints violated, but in doing so, some agents would prevent other agents from finding good value assignments. To deal with local optima, agents will have to go through a resolution process that will identify the sources of deadlocks and initiate actions for their resolution. However, resolution is not straightforward in distributed scenarios given the aforementioned restrictions. Therefore, agents have to rely on well crafted local mechanisms that implicitly result in the resolution of deadlocks. For example, in the Distributed Breakout (DBA) (Yokoo & Hirayama 1996) agents act unilaterally to resolve deadlocks, by placing weights on constraints and increasing weights on constraints causing deadlocks with other agents. In other algorithms such as the Asynchronous Weak Commitment Search (Yokoo & Hirayama 2000) and Asynchronous Backtracking (Yokoo *et al.* 1992) agents generate new constraints out of the deadlocks and as such, will avoid returning to those deadlocks as the search for a solution progresses. In algorithms like those presented in (Zhang *et al.* 2005) and (Liu, Jing, & Tang 2002), agents rely on stochastic mechanisms to avoid and escape from local optima without attempting to identify the causes of conflicts.

In this work, we introduce the Distributed Guided Local Search (Dist-GLS) algorithm for solving DisCSPs. In particular, we emphasise the strategies in the algorithm for dealing with local optima in the search for solutions. Our algorithm is partly an extension of the centralised Guided Local Search (GLS) (Voudouris 1997), with additional heuristics incorporated into it to enhance the algorithm's efficiency in distributed scenarios. The centralised GLS algorithm for search and combinatorial optimisation introduced structure-based learning for identifying the features of the solution that are particularly associated with local optima. Penalties are associated with these features, and these are incorporated into the objective function to be optimised. Penalties on features are increased when the underlying hill-climbing search hits a local optimum, if the features are present at that particular local optimum. The idea is to use penalties to change the shape of the objective landscape and as a result guide the search away from sub-optimal regions and to focus attention on more profitable areas of the search space.

The rest of this paper is structured as follows. In the next section we outline the general framework of Dist-GLS. Following that, we discuss the heuristics for resolving conflicts associated with local optima and justify their use. Next, we provide an illustration of the resolution process. Finally, we present results of preliminary experiments on random binary CSPs and graph colouring problems.

Distributed Guided Local Search

Dist-GLS is a semi-synchronous distributed iterative improvement algorithm. Aspects of the Distributed Agent Ordering (Hamadi 2002) algorithm are incorporated into Dist-GLS at the initialisation phase, which permits concurrent activity in unconnected parts of the problem. An agent's position in the ordering is also used to determine its priority within its neighbourhood. Each agent in the algorithm represents a single variable in the DisCSP, and is aware of the variable's domain and all the constraints attached to the variable. In addition, each agent also keeps track of penalties attached to individual domain values for its variable[1]. Each agent also maintains a no-good store used to keep track of deadlock situations encountered during the search process.

Agents take turns to choose values for their variables, and become active only after receiving choices of higher priority neighbours; then the individual agent will typically pick the value with the least sum of constraint violations and penalties (Figure 2, line 3). However, on detection of a quasi-local optimum i.e. in this case a deadlock situation where an agent is unable to find a value in its domain that reduces the number of constraints currently violated[2], the agent will initiate the conflict resolution process with lower priority neighbours (discussed in detail in the next section) and begin increasing penalties on particular domain values associated with this deadlock.

An agent may only communicate with other agents connected to it by constraints on their respective variables. To minimise the communication costs incurred during the search process, agents communicate with neighbours using a single message type that encapsulates both the values assigned to their variables and other messages. For example, to request that certain neighbours impose a temporary penalty on their current assignments, the message will be in this form: **message**(*its id*, *its value*, *addTempPenalty*).

Dealing with local optima

A two-tiered penalty system is introduced as opposed to the uniform penalty system adopted in GLS. This is implemented as follows:

1. On detection of a quasi-local optimum, an agent checks its no-good store initially to find out if the current inconsistent state had been previously visited. If it is not the case, then the agent imposes a temporary penalty on its current value and sends a message to all lower priority neighbours violating constraints with it requiring them to do the same (Figure 2, lines 7-11). The temporary penalty is discarded immediately after it is used.
2. If it is the case that a deadlock state has been previously encountered, then the agent increases the incremental penalty on its current domain value. In addition, the agent sends a message to all its lower priority neighbours requesting the same action (Figure 2, lines 13-16). If an agent receives requests for the imposition of a temporary penalty and increases of incremental penalties simultaneously from different neighbours, the latter takes precedence (Figure 3, lines 2-7).

[1] Domain values are used as problem features for the GLS aspect of the algorithm

[2] And all its neighbours values are unchanged from the previous iteration.

```
initialise

do
  when active
    evaluate state
    if penalty message received
      respond_to_message()
    else

      if current value is consistent
        reset incremental penalties
        send message(id, value, null) to neighbours
      else
        resolve_conflict()
      end if

    end if
  return to inactive state
until terminate
```

Figure 1: Dist-GLS: Main agent loop

The temporary penalty is a positive integer t ($t > 1$), and its value is problem dependent. In problems where the solution is a permutation of values a large t (e.g. t = 1000) works best, as it creates a perturbation large enough to allow agents find alternative values for their variables. While in problems involving some optimisation, like graph colouring, a small t (e.g. t = 3) is sufficient. The choice of temporary penalties in the first phase of the resolution process came out of empirical investigations into the effect of heuristics on conflict resolution. Results from those experiments indicate that while the use of temporal penalties resolves the least number of conflicts, it is unlikely to cause as many new constraint violations as with the use of incremental penalties on domain values. Therefore we concluded that temporary penalties would generally speed up the search process by helping the agents resolve conflicts without necessarily creating new conflicts in other parts of the problem.

All incremental penalties on domain values are set to zero at initialisation, and are increased incrementally by 1 in

```
procedure resolve_conflict()
  if neighbours state(t) ≠ neighbours state(t-1)
    select new value
    send message(id, value, null)
    return
  end if

  if neighbours state(t) is not in no-good store
    add neighbours state to no-good store
    impose temporary penalty on current value
    select new value
    send message(id, value, addTempPenalty)
  else

    if incremental penalty on current
     value < upper bound
      increase incremental penalty by 1
      select new value
    else
      select worst value in domain
    end if

      send message(id, value, increasePenalty)
  end if
end procedure
```

Figure 2: Dist-GLS: Resolve conflict

deadlock situations. In evaluating its options, an agent adds the incremental penalty of a domain value to the number of constraints violated by that value. An upper bound is imposed on the size of the incremental penalties. The intent is to prevent penalties increasing infinitely thereby exaggerating the effects on the objective landscape; and in addition, to avoid arrival at a situation where the penalties cease to contribute positively to the conflict resolution process. The upper bound is defined individually for each agent as *N*, where *N* is the number of neighbours. This number is chosen as the upper bound for each agent in order to incorporate structural aspects of the individual DisCSP in the algorithm and to avoid the need for parameter tuning. When the incremental penalties on the current domain value hit the upper bound, the agent uses the temporary constraint maximisation heuristic (TCM) (Fabiunke 2002) and selects the worst value in its domain. The aim is to perturb the agent's neighbourhood, forcing as many neighbours as possible to select alternative values for their respective variables.

Incremental penalties on all domain values are reset to zero whenever an agent finds a consistent assignment for its variable (Figure 1, line 9). Although there is a risk of losing experience gained in the search by doing so, this is an attempt to minimise the risk of the algorithm falling into a trap that causes it to oscillate between non-solution states. The idea of resetting penalties (or weights) has been explored in the literature, albeit in different contexts. For example, Morris (Morris 1995) argues that weight increases may conspire

```
procedure respond_to_message()
  if message is incremental incremental penalty
    if incremental penalty on current
     value < upper bound
      increase incremental penalty on current value
  else
    impose temporary penalty on current value
  end if
  select new value
  send message(id, value, null)
end procedure
```

Figure 3: Dist-GLS: Responding to penalty message received from higher priority agent

to block the path to a solution in the objective landscape. While, Voudouris (Voudouris 1997) points out that penalties may become invalid some point after they have been incurred. He therefore proposes, amongst other things, resetting penalties periodically. However, in this work, one intent of resetting penalties is to maintain algorithm robustness. Based on our argument that as agents do retain too much environmental history, the algorithm should be more responsive to dynamically changing problem specifications and communication failures while it runs.

An agent's no-good store is central to the coordination of its strategies for dealing with quasi-local-optima. The store is in some ways similar to tabu-lists (Glover 1989), serving as short term memory. The difference, however, is that the no-good store is not used to forbid the repetition of recent decisions by agents. A maximum of *N* recently encountered no-good states are maintained in the store on a First-In-First-Out basis. Unlike the incremental penalties, the no-good store is not emptied when the agent finds a consistent value for its variable.

Example of algorithm execution

A typical run of Dist-GLS is illustrated with the trivial DisCSP in Figure 4. In the example problem there are four agents, each representing one variable - **a**, **b**, **c**, and **d** respectively. Constraints on variable pairs are highlighted on the connecting arcs. The domains of the variables are shown in the figure (D_a, D_b, D_c, and D_d) as well as the number of constraints violated for each domain value (V_a, V_b, V_c, and V_d) given the current assignments to all variables. For example, both domain values for **b** violate the constraint (**b** = **c**) for the current assignment **c** = 0. The incremental penalties on the respective domain values are also displayed (P_a, P_b, P_c, and P_d).

The state of the DisCSP in Figure 4 is a deadlock because both agents **b** and **c** are at a quasi-local optima, each without possible improvements for their respective variables. At this stage, in Dist-GLS, agent **b** imposes a temporary penalty (*t* = 100) on its current value (see Figure 5 (i)). The violations for **b**'s current value are increased prompting it to change its assignment to **b** = 2. At the same time, agent **b** sends

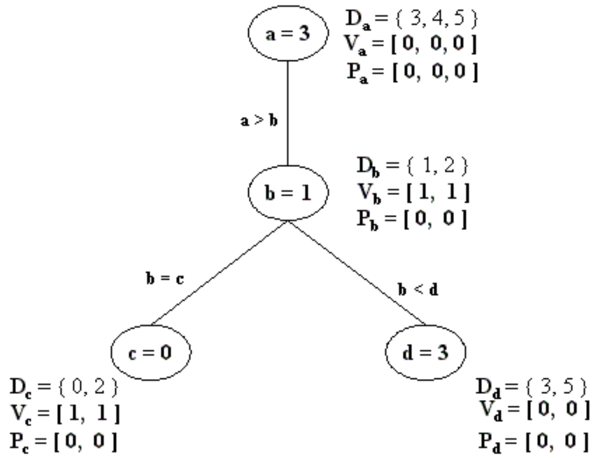


Figure 4: Example DisCSP.

its new value and a temporary penalty message to lower priority agent **c** (message(*b*, 2, *addTempPenalty*)) and sends its other neighbours an update of its new assignment (message(*b*, 2, *null*)). Agent **b** also places the current values of all its neighbours in its no-good store e.g. **no-good**(**a**=3, **c**=0, **d**=3). The no-good is not counted as a new constraint, it is only referred to if the conflict is not resolved using the temporary penalties. In Figure 5 (ii), agent **c** receives the temporary penalty message and imposes a temporary penalty on its current value, forcing it to change the value assigned to its variable. In this trivial example, the conflict is resolved with temporary penalties in the first phase of conflict resolution. If it is the case that conflicts remain unresolved after this phase, agent **b** will initiate subsequent phases of the resolution process going through incremental penalty increases and resorting to the TCM heuristic as a last resort, as discussed in the previous section.

Empirical Evaluation

To evaluate the performance of Dist-GLS, it was tested with randomly generated problems. The algorithm's performance was evaluated along two criteria; (1) effectiveness in terms of the ability to find solutions to problems, and (2) efficiency with respect to the number of iterations utilised in finding solutions. We use the number of iterations, rather than the run times, to abstract out implementation and environmental influences on the behaviour of the algorithm. We also compare its performance with DBA on the same set of problems.

We use DBA for comparison because it is a widely accepted benchmark with which incomplete iterative improvement algorithms are compared. In addition, we also chose it, over other distributed algorithms, because it has a somewhat similar approach with Dist-GLS for dealing with local optima. To verify our implementation of DBA it was first tested on graphs generated with the same method used in (Yokoo & Hirayama 1996) and the results are at least as good as those reported in that work (see Table 1).

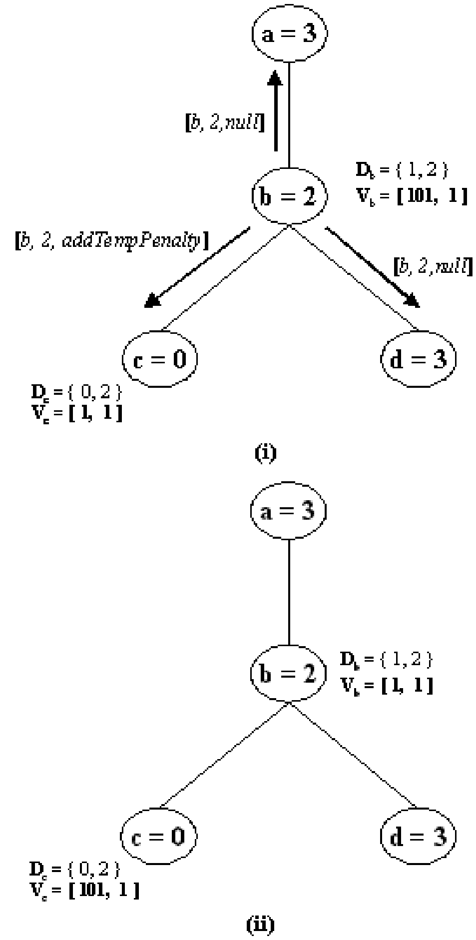


Figure 5: Example of Dist-GLS execution.

Random Binary Constraint Satisfaction Problems

Dist-GLS was first tested on a class of random constraint satisfaction problems, strictly composed of binary relational constraints[3] i.e. the constraints are relational operators (e.g. $=$, $\neq$, $<$, $>$) between pairs of variables. The problems were generated using the standard Model B (Palmer 1985) and the individual constraints were built around support values for variables which ensured that each problem was solvable, having at least one solution.

In the results plotted in Figures 6 (a) and (b), each algorithm was tested on 500 problems for each constraint density. There were 70 variables in each problem with five values in each variable's domain. A time limit of *100n* iterations was imposed on both algorithms after which an attempt was reported as failure. A large temporary penalty (*t* = 1000) was used for Dist-GLS in these experiments. The plots summarise the results of experiments studying the behaviour of both algorithms on the relational CSPs as constraint density increases. The first plot suggests, first of all,

[3] For the rest of this paper these are referred to as Relational CSPs.

n	$m = n * 2$ (a)	(b)	$m = n * 2.7$ (a)	(b)
90	150	*128*	517	*478*
120	210	*152*	866	*836*
150	278	*168*	1175	*1173*

Table 1: Average search costs of solving random graph colouring problems with our implementation of DBA (b), compared to results of similar experiments reported in Yokoo and Hirayama (1996) (a). In these experiments we tested on 100 graphs (k=3) for each n (number of nodes) and m (number of edges).

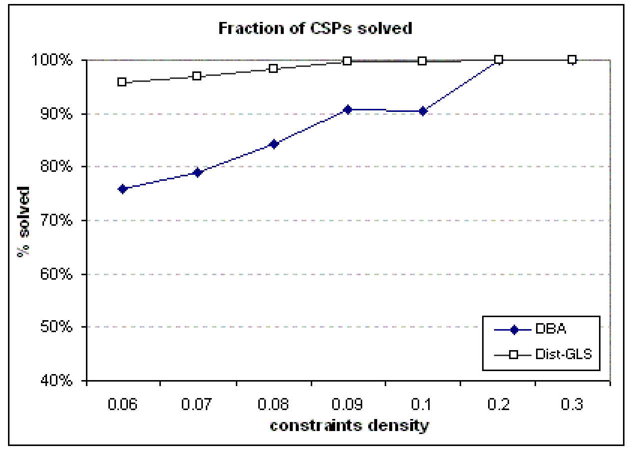
that problems with low constraint densities (less than 0.2) appear to be more difficult than those with higher constraint densities. This is drawn from the fact that both algorithms solved fewer problems in this region. More importantly, the figures also show that Dist-GLS found more solutions in the region of difficult problems and consistently required fewer iterations than DBA in finding the solutions. However, the figures understate the advantage of Dist-GLS over DBA. This is because for every iteration in DBA, each agent sends two messages (*improve* and *ok?* messages) as opposed to the single message used in the Dist-GLS. Therefore, the actual cost incurred by DBA is twice that displayed.

Graph Colouring

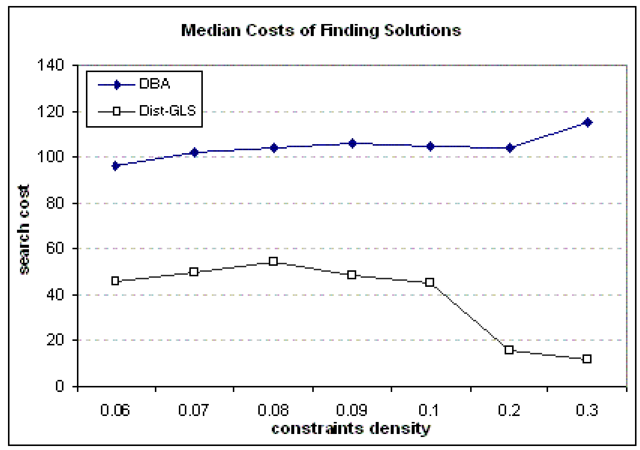
Performance of Dist-GLS was also evaluated on the distributed version of the graph colouring problem. In this case, both algorithms (Dist-GLS and DBA) were tested on random 100-node graphs with varying degrees of connectivity. The graphs were generated using the algorithm suggested in (Fitzpatrick & Meertens 2002).

Given that graph colouring is a hybrid of constraint satisfaction and optimisation, Dist-GLS was modified slightly to take into account the optimisation requirements of the problem. Therefore, agents were forced to choose the leftmost minimum value in their domains when two or more values in the domains had equal number of minimal constraint violations; as we sought to find a solution with the least number of colours. This strategy was proposed in Liu et al's work (Liu, Jing, & Tang 2002). The heuristic previously employed on the binary CSPs required agents to retain their current values when faced with more than one equally appealing option [4]. In addition, we used a small temporary penalty ($t = 3$).

Results of experiments using the modified version of the algorithm, as well as the comparison with DBA, are plotted in Figure 7. In these experiments, each algorithm was tested on 100 random graphs for each degree (i.e. the average number of edges per node), with an upper bound of 10,000 iterations before an attempt was recorded as a failure. The plot in Figure 7(a) shows that both algorithms were able to solve a high percentage of problems, deteriorating slightly for DBA in the phase transition region. However, the plot of median search costs in Figure 7(b) shows a significant difference in performance in favour of Dist-GLS. Results in the plot show



(a)



(b)

Figure 6: Comparison of Dist-GLS and DBA behaviour on relational CSPs (n=70, d=5) in terms of: **(a)** number of problems solved and **(b)** median search costs.

that in the majority of cases the number of iterations required by DBA to find solutions to problems is twice that of Dist-GLS.

Completeness and Termination Detection

The Distributed Guided Local Search algorithm is not complete and, therefore, will not terminate for unsolvable problem instances. As evident in the results presented, it solved a high percentage of problems even though it is not guaranteed to find a solution within reasonable time; as with all incomplete algorithms. Since there is no inbuilt termination detection mechanism within the algorithm, we have relied on a 'global overseer' to detect global termination states in the experiments performed so far. To compensate for this in the comparison with DBA, we deducted, from the results of DBA, the number of iterations deemed to be utilized mainly for termination detection. However, we expect that since Dist-GLS is largely synchronous, the termination mechanism required will not necessarily be as complicated as that

[4] Assuming the agent's current value is equally as appealing

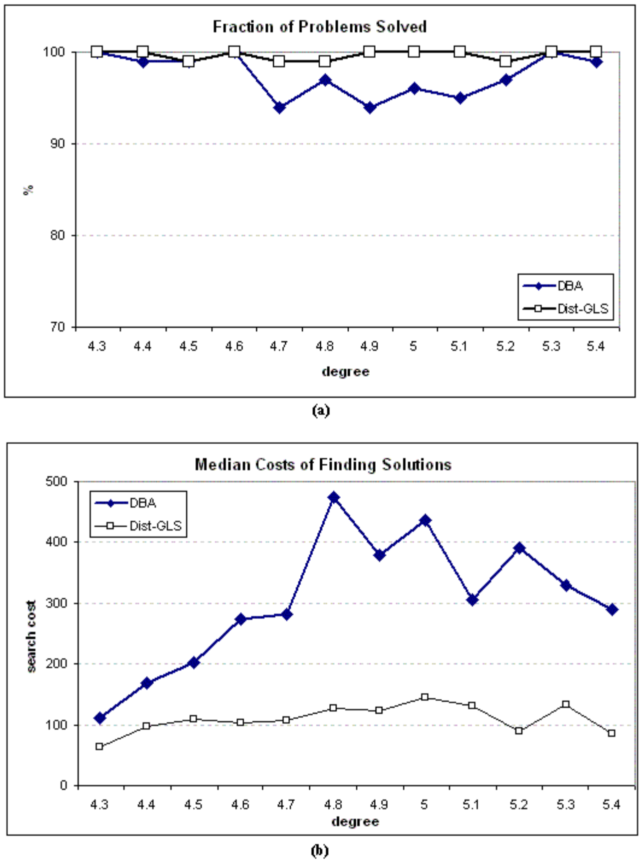


Figure 7: Phase transition behaviour of Dist-GLS and DBA on graph colouring problems (n = 100, k = 3) as average node connectivity increases. Algorithms are compared with respect to: **(a)** number of problems solved and **(b)** the median search cost.

used in asynchronous algorithms (like DBA), and therefore may not require as many iterations for correct termination.

Conclusions

In this work, we have introduced the Distributed Guided Local Search algorithm. The algorithm is partly based on the centralised Guided Local Search algorithm for combinatorial optimisation problems, with additional heuristics incorporated into it to improve its efficiency. These heuristics and the justifications for their inclusion were discussed in the paper. In addition, we presented the results of empirical evaluations of Dist-GLS on two problem classes, as well as a comparison with DBA. To summarise, the results show that on average the cost of finding solutions to problems with Dist-GLS is about half that of DBA. By implication, offering significant savings in communication costs incurred in the process of solving distributed CSPs. The results, whilst still preliminary, also show that Dist-GLS finds more solutions to problems than DBA especially in the region of difficult problems.

Our future work with Dist-GLS aims to address some of the issues highlighted here, especially the need to incorporate a termination detection mechanism into it. We have tested our algorithm on two problem classes and we intend to continue investigations into its performance on different problem classes such as non-binary constraint satisfaction problems. We will also consider extensions to the algorithm to enable it solve DisCSPs where agents represent multiple local variables.

References

Fabiunke, M. 2002. A swarm intelligence approach to constraint satisfaction. In *Proceedings of the conference on Integrated Design and Process Technology IDPT 2002*, CD–ROM, 4 pages. Society for Design and Process Science.

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. 1989. Tabu search - part i. *ORSA Journal on Computing* 1(3):190–206.

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

Liu, J.; Jing, H.; and Tang, Y. Y. 2002. Multi-agent constraint satisfaction. *Artificial Intelligence* 136(1):101–144.

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.

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.