DisBO-wd: a Distributed Constraint Satisfaction Algorithm for Coarse-Grained Distributed Problems

Muhammed Basharu[1], Inés Arana[2], and Hatem Ahriz[2]

[1] 4C, University College
Cork, Ireland
[2] School of Computing
The Robert Gordon University
Aberdeen, UK
mb@4c.ucc.ie, {ia,ha}@comp.rgu.ac.uk

Abstract. We present a distributed iterative improvement algorithm for solving coarse-grained distributed constraint satisfaction problems (DisCSPs). Our algorithm is inspired by the Distributed Breakout for coarse-grained DisCSPs where we introduce a constraint weight decay and a constraint weight learning mechanism in order to escape local optima. We also introduce some randomisation in order to give the search a better chance of finding the right path to a solution. We show that these mechanisms improve the performance of the algorithm considerably and make it competitive with respect to other algorithms.

1 Introduction

The recent growth of distributed computing has created more opportunities for collaboration between agents (individuals, organisations and computer programs) where there is a shared objective but, at the same time, there is also a competition for resources. Hence, participants make compromises in order to reach agreement - a process which can be automated if the situation is modelled as a Distributed Constraint Satisfaction Problem (DisCSP) [12]. DisCSPs formally describe distributed problems where each participant in the problem is represented by an agent, and the collection of agents have to collaborate in order to reach a satisfactory agreement (or find a solution) for a problem. Research in this emerging field includes problem solving techniques which are classified as constructive search or iterative improvement search. Iterative improvement search is normally able to converge quicker than constructive search on large problems, but it has a propensity to converge to local optima. Previous work on iterative improvement search has considered a variety of techniques for dealing with local optima. Prominent amongst these is the breakout, which attaches weights to constraints which are difficult to satisfy [13].

Distributed iterative improvement algorithms for DisCSPs such as DisPeL[1] and DBA[13] assume that each agent is responsible for one variable only and knows its domain, the constraints which apply to the variable, the agents whose variables are constrained with its variable and the current value for any varible directly related to

its own variable. In other words, agents are not allowed to be responsible for whole subproblems, but just a single variable and the problem is said to be *fine-grained*. As a result, there is a very limited amount of computation that agents can perform locally and all the search effort is focused on the distributed collaborative activity which is expensive. In constrast, if agents are allowed to own whole subproblems (i.e. the problem is *coarse-grained*) agents are able to carry out a substantial amount of computation locally.

This paper presents DisBO-wd, an iterative distributed algorithm for solving coarse-grained DisCSPs which uses weights on constraints in order to escape local optima. These weights are continuously decayed throughout the problem solving and increased whenever constraints are not satisfied. Empirical results show that DisBO-wd is effective, solving most problems in reasonable time and at a lower cost than other algorithms.

The remainder of this paper is structured as follows. Section 2 gives some definitions and explains related research. Next, DisBO-wd is introduced in Section 3. Finally, in Section 4, we present the results of empirical evaluations of DisBO-wd along with comparisons with other similar algorithms.

2 Background

A constraint satisfaction problem (CSP) is a triple $< V, D, C >$ where V is a set of variables, D is a set of domains (one per variable) and C is a set of constraints which restrict the values that variables can take simultaneously. Two variables are said to be *neighbours* if they both participate in the same constraint. Some algorithms for solving CSPs attach a *priority* to each variable in order to rank the variables from high to low priority.

A DisCSP is a CSP where the variables, domains and constraints are distributed over a number of agents and whose details cannot be centralised for a number of reasons such as privacy, security and cost. These problems are, therefore, solved by agent-based distributed algorithms which collaborate in order to find a solution. Some algorithms for solving CSPs attach a *priority* to each agent in order to rank the agents from high to low priority, eg. [14].

When DisCSPs are made up of interconnected subproblems which are naturally distinct from other subproblems, each individual subproblem is a CSP with its own set of variables and constraints between those variables, as well as constraints between some variables in the local CSP and variables in other sub-problems (as illustrated in Figure 1). Therefore, rather than representing one variable, each agent in the DisCSP represents a sub-problem, i.e the DisCSP is *coarse-grained*. For example, distributed university timetabling is a coarse grained DisCSP, where agents represent lecturers and each sub-problem is the set of courses taught by an individual. The constraints in the local subproblems (CSPs) include that an individual cannot teach two different courses at the same time (intra-agent constraints), as well as constraints to prevent some clashes with courses taught by other lecturers (inter-agent constraints) either because of student course registrations or resource availability.

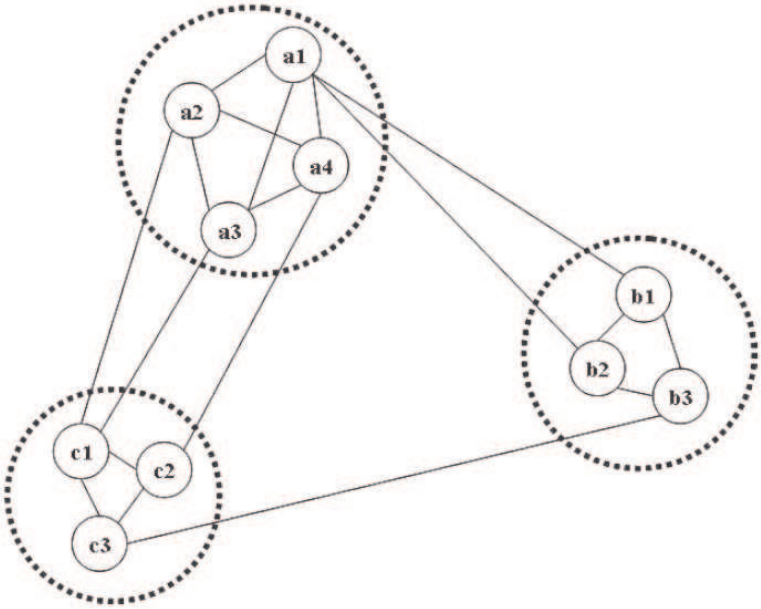

Fig. 1. An illustrative example of a coarse-grained DisCSP, with 3 inter-connected sub-problems/agents.

Agents in coarse-grained DisCSPs (with more than one variable per agent) are more complex than fine-grained DisCSP agents (with only one varible per agent since they have much more problem information available and, therefore, can carry out more local inference in the search for a solution. The amount of local computation (and the level of granularity) required by complex agents can vary along two extremes [14], ranging from fine-grained to coarse-grained DisCSPs. If a problem is implemented as a fine-grained DisCSP, each of its subproblems acts as a variable. Therefore, each agent does all its local computation before hand by finding all possible solutions for its subproblem which are taken as the "domain values" of subproblem variable, e.g. [11]. Unfortunately, when local sub-problems are large and complex, it may be impossible or impractical to find all local solutions.

At the other extreme, each variable in a subproblem can be seen as a virtual agent, i.e. a single variable per agent and, therefore, local computation is minimal and all effort is expended on the distributed search. Agents simulate all activities of these virtual agents including communications between them and other real agents. This approach does not take full advantage of local knowledge and, therefore, it is costly since the expense of local computation is significantly lower than that of communications between virtual agents. It appears that an approach where there is a balance between the two extremes could be beneficial, so that agents can enjoy the flexibility of the finest level of granularity and at same time exploit the clustering of local variables to speed up the search.

In distributed backtracking algorithms for DisCSPs with complex agents, most prominently Asynchronous Backtracking [6, 8] and Asynchronous Weak Commitment Search [14], the granularity of the single variable per agent case is still maintained and, therefore, variables are inadvertently treated as virtual agents; especially since these algorithms are direct extensions of earlier versions for fine grained DisCSPs. By taking each variable as a virtual agent, agents in those algorithms use a single strategy to deal with both inter-agent and intra-agent constraints as there is typically no distinction between the constraints. As such, deadlocks (local optima) are still detected (and no-goods generated) from each variable rather than from entire sub-problems. At the same time, the amount of computation done locally within agents is still significant. Each agent will typically try, exhaustively in the worst case, to find a local solution that is consistent with higher priority external variables before either extending the partial solution or requesting revision of earlier choices by other agents.

3 DisBO-wd

3.1 DBA, Multi-DB and DisBO

The distributed breakout algorithm (DBA) [13] is an iterative improvement DisCSP algorithm which uses a single variable per agent. In DBA, agents carry out a distributed steepest descent search by exchanging possible improvements to a candidate solution and implementing the best improvements that do not conflict with each other. Agents act concurrently alternating between the *improve* and *ok?* cycles. In the *improve* cycle, each agent finds the value in its domain that minimises its weighted constraint violations and computes the improvement to its current assignment. These improvements are exchanged between the agents. In the *ok?* cycle the agents with the best improvements are allowed to change their values. Ties are broken with the agents lexicographic IDs in the case that two or more neighbouring agents have the same possible improvement.

In order to escape local optima, DBA attaches a weight to each constraint which is increased whenever the constraint is violated at a quasi-local-optimum, i.e. a state in which a subset of connected agents cannot find any improvements to their local evaluations. Empirical evaluation of DBA showed that it outperformed the Asynchronous Weak Commitment Search on difficult problem instances; DBA solved more problems and it did so in less time [13].

Multi-DB [4, 5] is an extension of DBA for coarse-grained distributed SAT problems. DisBO [2] is another extension for coarse-grained distributed project scheduling and graph colouring. This version was largely based on DBA's framework but differed in its emphasis on increasing weights only at *real* local optima. DisBO differs from Multi-DB in that it has an additional third cycle for global state detection, since weights are only increased when the search is stuck at real local optima and not at quasi-local-optima. Therefore, in addition to the *improve* and *ok?* cycles, there is a *detect-global-state* cycle, which is used to determine that either a solution has been found, that the maximum number of cycles has been reached, or that the search is stuck at a real local optimum. But, the *detect-global-state* cycle is expensive, in terms of messages sent, because it requires agents to continuously exchange state messages

until they have determined that all messages have reached all agents in the network. This was needed to get a snapshot of the state of the entire network without resorting to a global broadcast mechanism where each agent is assumed to know every other agent in the network.

DisBO limits the amount of computation done locally within each agent to allow agents to focus on the collaborative aspect of the problem solving activity. In DisBO, each agent's variables are partitioned into two sets, private and public variables. The private variables are those variables that have no inter-agent constraints attached to them. The bulk of the local computation done by agents in DisBO are with these private variables, where in each improvement phase the agents repeatedly select values for these variables that minimise the weighted constraint violations until no further improvements are possible. The public variables, on the other hand, are treated like virtual agents and DBA's coordination heuristic is used to prevent any two public variables (even those within one agent) from changing their values simultaneously unless the concurrent changes do not cause the constraints between them to be violated.

3.2 DisBO-wd

DisBO-wd is our DisCSP algorithm for coarse-grained CSPs based on DisBO where the weight update scheme is replaced with a weight decay scheme inspired on Frank's work on SAT solving with local search [3]. Instead of modifying weights only when a search is stuck at local optima, weights on violated constraints are continuously updated after each move. At the same time, weights are also decayed at a fixed rate during the updates to allow the algorithm focus on recent increments. Frank argues that this strategy allows weights to provide immediate feedback to the variable selection heuristic and hence emphasise those variables in unsatisfied clauses. We modified the update rule further, so that weights on satisfied constraints are continuously decayed as well. Therefore, before computing possible improvements, in DisBO-wd, agents update their constraint weights as follows:

Weights on violated constraints at time t are computed as

$$W_{i,t} = (dr * W_{i,t-1}) + lr$$

Weights on satisfied constraints at time t are decayed as

$$W_{i,t} = max((dr * W_{i,t-1}), 1)$$

where:
dr is the decay rate ($dr < 1$).
lr is the learning rate ($lr > 0$).

From empirical investigations (see section 3.4), we found that DisBO-wd's performance was optimal with the parameters set to $dr = 0.99$ and $lr = 8$. With the new weight update scheme, we were able to reduce the number of DisBO's cycles from three to two, since it was no longer necessary to determine if the search was stuck at real local optima - a big cost saving. We also moved the termination detection

mechanism into the *ok?* cycle, as in the original distributed breakout framework. Other modifications such as probabilistic weight resets and probabilistic weight smoothing [7] were considered, but found to be weaker than our new weight decay mechanishm.

In DisBO when two neighbouring variables have the same improvement, the variable with the lower lexicographic ID is deterministically given priority to make its change. We have replaced this coordination heuristic with a random break [10] so in each improvement cycle agents select and communicate random tie-breaking numbers for each variable, and when there is a tie the variable with the lower number is given priority.

3.3 Creating coarse-grained DisCSPs

In order to obtain coarse-grained DisCSPs, publicly available problem instances from CSPLib[1] and SATLib[2], as well as randomly generated problems, were partitioned into evenly sized inter-connected sub-problems using a simple partitioning algorithm which ensured that the cluster of variables within each agent were meaningful, i.e. there are constraints between the variables belonging to an agent. For each agent (a_i) its subproblem was created as follows:

1. A randomly selected variable (x_i) that is not already allocated to another agent is allocated to a_i.
2. A variable constrained with x_i is randomly selected and allocated to a_i.
3. The process of randomly selecting one of the variables already allocated to a_i and selecting a random neighbour of the variable for allocation to a_i is repeated until the number of required variables for a_i have been found.
4. With a small probability (p), a randomly selected variable is allocated to a_i, even if it is not connected with any of a_i's existing variables.

3.4 Determining optimal parameter values for DisBO-wd

We have replaced the weight update mechanism of DisBO with the scheme for continuous weight updates proposed in [3]. This new scheme introduces two new parameters into DisBO-wd i.e. the *learning rate (lr)* and the *decay rate (dr)*. The *learning rate* controls how fast weights on violated constraints grow in DisBO-wd, while the *decay rate* biases the search towards the most recent weight increases.

In his work on SAT solving with a modified GSAT [9] algorithm, Frank [3] found that the decay rate was optimal at $dr = 0.999$, more problems were solved within an allotted time than with the value set to 0.95 and 0.99. He also found that the learning rate was optimal at $lr = 1$ compared to runs with the values 8, 16, and 24. However, DisBO-wd differs from GSAT in many respects especially given the amount concurrent changes that take place in distributed search. Therefore, we had to carry out an experiment to determine optimal values for the parameters in the distributed algorithm. We used distributed SAT instances and random DisCSPs, evaluating DisBOs performance on 100 instances in each case, with the parameters set to

[1] www.csplib.org [accessed 26 March 2007].
[2] www.satlib.org [accessed 26 March 2007].

$lr \in \{1, 2, 3, 5, 8, 10, 12, 16\}$ and $dr \in \{0.9, 0.95, 0.98, 0.99\}$. In Tables 1 and 2, we summarise the results from this experiment, showing the percentage of problems solved, the average and the median search costs incurred where we limited DisBO-wd to 10,000 iterations on each attempt on the SAT problems and 12,000 iterations on each attempt on the random DisCSPs.

		SAT Problems			**Random Problems**		
		%	average	median	%	average	median
dr	lr	solved	cost	cost	solved	cost	cost
0.9	1	60	85	28	29	1598	462
	2	82	178	90	79	1160	550
	3	84	151	108	95	2298	1458
	5	94	303	141	99	2412	1648
	8	88	273	139	100	2416	1658
	10	92	300	129	96	2472	1876.5
	12	92	233	176	93	2080	1606
	16	88	314	145	94	2260	1286.5
0.95	1	88	208	111	87	1304	663
	2	98	304	144	100	1986	1030.5
	3	100	303	130	100	2226	1414.5
	5	100	329	195	98	1952	1114
	8	100	259	161	93	1922	1238
	10	98	338	181	95	1876	1104
	12	100	314	136	96	2048	1232
	16	100	358	233	100	2050	1182.5

Table 1. Performance of DisBO-wd on Distributed SAT problems and Distributed Random problems for $dr \in \{0.9, 0.95\}$ and variable lr.

The results in Tables 1 and 2 summarise attempts to solve 50-literal SAT instances from the SATLib problem set. As dr increases, DisBO-wd solved more problems but there is no clear relationship between the search costs and the decay rate. Given these results we chose $dr = 0.99$ and $lr = 1$.

Tables 1 and 2 also show the results of 100 runs for random problems with <number of variables $n = 60$, number of values in a domain $d = 10$, constraint density $p_1 = 0.1$, constraint tightness $p_2 = 0.5$ >. The search cost is better for high values of dr, suggesting that the search benefits from retaining some information of not too recent weight increases for as long as possible and they are not quickly dominated by newer weight increases. However, it appears that the learning rate lr has a different effect on performance in this domain. The algorithm generally does not fare too well with the smallest and largest values for this parameter. The results, although not clear cut, show that DisBO-wd is optimal with the values 3,8, or 10 (at $dr = 0.99$), where the average search costs are minimal and the percentage of problems solved are significantly high. But, we arbitrarily chose $lr = 8$ and $dr = 0.99$ for the experiments

		SAT Problems			Random Problems		
dr	lr	% solved	average cost	median cost	% solved	average cost	median cost
0.98	1	100	236	119	95	1834	1163
	2	100	375	198	98	1590	1050
	3	100	247	174	96	1568	916
	5	100	263	213	99	1822	1024
	8	100	286	202	99	2114	1038
	10	100	439	176	96	1476	984
	12	100	380	219	100	1752	1082.5
	16	100	386	174	98	2018	1152
0.99	1	100	186	130	97	1668	978
	2	100	252	127	97	1748	1120
	3	100	243	189	97	1506	832
	5	100	235	139	93	1528	826
	8	100	373	235	99	1554	858
	10	100	312	213	100	1682	828.5
	12	100	318	207	96	2152	1432.5
	16	100	269	213	97	2454	1376

Table 2. Performance of DisBO-wd on Distributed SAT problems and Distributed Random problems for $dr \in \{0.98, 0.99\}$ and variable lr.

with the algorithm because it solved slightly more problems than with $lr = 3$ and the search costs were lower than with $lr = 10$.

3.5 DisBO vs. DisBO-wd

A series of experiments were conducted in order to analyse the effect of the various modifications to DisBO. Thus, DisBO and DisBO-wd were compared by evaluating the results of running them on the same problems. Table 3 summarises results from experiments on critically difficult distributed graph colouring problems with 10 variables per agent and $< number\ of\ colours\ k = 3, degree = 4.7 >$ - for an explanation of how these problems were generated see section 3.3. The results show that DisBO-wd solved more problems than DisBO, especially on the larger problems. Furthermore, DisBO-wd required fewer cycles to solve the problems.

Table 4 contains results of experiments on randomly generated problems with $< domain\ size = 10, density\ (p1) = 3n, tightness\ (p2) = 0.5, number\ of\ iterations = 100 * n >$ where n is the number of variables. The results show that, like with graph colouring problems, DisBO-wd solved substantially more problems than DisBO and required fewer cycles.

	% solved		median cost	
num. vars.	DisBO	DisBO-wd	DisBO	DisBO-wd
50	100	100	222	115
60	100	100	366	199
70	98	100	480	249
80	98	100	741	27
90	99	100	1095	536
100	95	100	2121	723
110	92	100	1257	655
120	86	100	2214	1011
130	79	98	2400	1354
140	78	100	3755	1534
150	78	100	4929	2086

Table 3. DisBO vs. DisBO-wd on random distributed graph colouring problems of various sizes. Each point represents attempts on 100 problems.

		% solved		median cost	
num. vars.	num. agents	DisBO	DisBO-wd	DisBO	DisBO-wd
50	5	61	100	474	402
	10	66	100	466	512
100	5	56	100	476	563
	10	58	100	975	569

Table 4. DisBO vs. DisBO-wd on random problems of various sizes. Each point represents attempts on 100 problems.

4 Empirical Evaluation

An experimental evaluation of DisBO-wd was carried out using coarse grained versions of several DisCSPs including boolean satisfiability formulae (SAT) and randomly generated DisCSPs. In each case, the algorithm's performance was compared to the Asynchronous Weak Commitment Search algorithm (Multi-AWCS)[14] and, in the case of SAT problems, to Multi-DB. These two algorithms were selected because they are, to our knowledge, the only distributed local search algorithms which allow more than one variable per agent. Note that Multi-DB was not used with random problems since it was not designed to solve these. The algorithms were compared on the percentage of problems solved within a maximum number of iterations (or cycles)[3]. The number of iterations (cycles) was used as the measure of efficiency - a widely used measure for distributed iterative improvement algorithms since it is machine independent and, in the case of synchronous algorithms, the number of messages exchanged between agents and the number of constraint checks can be inferred or approximated with this metric.

[3] Given its completeness and unlimited time, Multi-AWCS is guaranteed to solve all problems used since they all have solutions. But, in this case we are interested in its performance in bounded time.

Unlike the breakout-based algorithms, Multi-AWCS is a complete algorithm and is not built around the idea of resolving deadlocks by increasing constraint weights. Rather, it combines backtracking and iterative improvement search and deals with local optima through a combination of variable re-ordering and storage of explicit no-goods. Although this algorithm has been shown to outperform other distributed backtracking algorithms [14], it can require an exponential amount of memory to store no-goods.

4.1 Distributed SAT problems

We evaluated the performance of DisBO-wd and the benchmark algorithms on distributed SAT problems. Satisfiable 3-SAT instances from the SATLib dataset made up of formulae with 100, 125 and 150 literals were used for the experiments. These were transformed into coarse-grained DisCSPs with the technique specified in Section 3.3. We did not run any experiments with Multi-DB and Multi-AWCS, rather we used results on experiments with the same instances from [4], published by the algorithms' authors, as benchmarks[4]. Note that the results for Multi-DB are for a version with periodic random restarts, which its authors found solved more problems than the original version [4] and that the version of Multi-AWCS which they used has no no-good learning to keep their comparisons with Multi-DB fair. DisBO-wd was run once on each instance, and was limited to $100n$ iterations (where n is the number of literals in a formulae) before attempts were recorded as unsuccessful. Note that in the experiments reported in [4], Multi-DB and Multi-AWCS were limited to a maximum of $250n$ iterations on their runs and, therefore, we are giving our own algorithm less time to attempt to solve the problem. The results in Tables 5, 6, and 7 show the percentage of problems solved and the average and median search costs for the problems which were successfully solved.

It can be seen from the results that DisBO-wd generally performed substantially better than the other 2 algorithms. Its average and median costs are significantly better than those of Multi-DB and its performance is at least as good with the following two exceptions: (i) SAT problems with 100 literals where the algorithm has only 2 agents - DisBO-wd's cost was higher; (ii) SAT problems with 150 literals where the algorithm has 3 or 5 agents - the percentage of problems solved by DisBO-wd was marginally lower and the median cost with 3 agents was higher. Also note that DisBO-wd has a consistency in its search costs that Multi-DB does not match. For example, average search costs in the 150 literal problems increase by about 350% as the number of agents increase for Multi-DB while DisBO-wd's average search cost remains within a 20% range of the minimum average without a clear degradation in performance as the number of agents increase. While both DisBO-wd and Multi-DB, rely on modifying constraint weights to deal with deadlocks, DisBO-wd is less affected by the distribution of variables to agents.

With respect to Multi-AWCS, DisBO's performance was significantly better in all cases. Multi-AWCS solved the least number of problems and it had the highest search costs.

[4] Variables are randomly distributed amongst agents in [4], so from each agent's perspective the problems may not be exactly the same.

algorithm	agents	% solved	average cost	median cost
Multi-DB	2	99.9	886	346
	4	100	1390	510
	5	100	1640	570
	10	99.6	3230	1150
	20	99.7	3480	1390
Multi-AWCS	2	99.9	1390	436
	4	98.7	4690	1330
	5	97.6	6100	1730
	10	96.8	7630	2270
	20	95.0	8490	2680
DisBO-wd	2	100	923	515
	4	100	948	495
	5	100	984	490
	10	99.9	1003	516
	20	99.8	993	510

Table 5. Performance of DisBO-wd and other algorithms on 1000 random distributed SAT problems with 100 literals distributed evenly amongst different numbers of agents.

algorithm	agents	% solved	average cost	median cost
Multi-DB	5	100	2540	816
	25	100	6300	2330
Multi-AWCS	5	87	19200	9290
	25	80	25500	15800
DisBO-wd	5	100	1727	725
	25	100	1686	921

Table 6. Performance of DisBO-wd and other algorithms on 100 random distributed 125 literal SAT problems.

4.2 Random distributed constraint satisfaction problems

We evaluated the algorithms performance on random DisCSPs. In this experiment Multi-AWCS only produces results in the runs with the smallest sized problems. It is well documented (for example in [8]) that Multi-AWCS can require an exponential amount of memory to store no-goods during an attempt to solve a problem. The number of no-goods generated can increase exponentially on large problems, and since each no-good may be evaluated at least once in each iteration, the length of time to complete each iteration increases dramatically as the search progresses. In our experience with Multi-AWCS, we found that it typically ran out of memory on runs with large problems, especially for DisCSPs with 60 or more variables and the algorithm sometimes required considerable amounts of time to solve even a single instance. We used three groups of problems with varying sizes and 100 problems in each group. The results of these experiments are summarised in Table 8 where we show the percentage

algorithm	agents	% solved	average cost	median cost
Multi-DB	3	100	2180	608
	5	100	3230	1200
	10	96	9030	2090
	15	98	9850	3850
Multi-AWCS	3	81	24300	11100
	5	67	37100	26100
	10	61	39400	36000
	15	61	42300	41700
DisBO-wd	3	99	2078	874
	5	99	2186	910
	10	99	2054	1012
	15	98	1893	898

Table 7. Performance of DisBO-wd and other algorithms on 100 random distributed 150 literal SAT problems.

of problems solved, and the average and the median iterations from successful runs on attempts on 100 instances for each problem size.

Multi-AWCS has lower search costs than DisBO-wd for problems with 50 variables when 5 or 10 agents are employed. It also solves slightly more problems when 5 agents are used. However, Multi-AWCS was unable to return results for problems with 100 and 200 variables, regardless of the number of agents used. DisBO-wd gave results for all problem sizes, although its performance degraded considerably on the largest problems.

algorithm	n	agents	% solved	average cost	median cost.
Multi-	50	5	100	738	288
AWCS		10	98	995	527
	100	5	out	of	memory
		10	out	of	memory
	200	5	out	of	memory
		10	out	of	memory
		20	out	of	memory
DisBO-	50	5	94	1927	1336
wd		10	99	1855	1104
	100	5	83	4996	2922
		10	88	4695	3065
	200	5	62	13454	8060
		10	65	16832	14432
		20	57	13289	9544

Table 8. Performance of algorithms on random DisCSPs ($< n$, domain size $d = 10$, constraint density $p1 \approx 0.1$, constraint tightness $p2 = 0.5 >$).

5 Discussion and conclusions

We have presented DisBO-wd, a distributed iterative improvement algorithm for solving coarse-grained DisCSPs which employs the breakout technique in order to escape local optima. Unlike other similar algorithms, DisBO-wd uses a weight decay and a learning rate in order to control constraint weights. In addition, its agent coordination strategy is non-deterministic, since it contains a stochastic mechanism for tie-breaking when more than one agent offers the best improvement.

DisBO-wd is competitive with respect to Multi-DB, which is the other algorithm that relies on constraint weights to deal with local optima; but unlike DisBO-wd, weights in Multi-DB are allowed to grow unbounded. DisBO-wd's search costs were generally substantially lower than those for Multi-DB in the SAT problems. DisBO-wd was also significantly better than Multi-AWCS in all the experiments with SAT problems. With random problems, Multi-AWCS produced better results for small problems with 50 variables but DisBO-wd was able to return results for large problems (with 100 and 200 problems) which Multi-AWCS was unable to solve due to its memory requirements.

References

1. Muhammed Basharu, Inés Arana, and Hatem Ahriz. Solving DisCSPs with penalty-driven search. In *Proceedings of AAAI 2005 - the Twentieth National Conference of Artificial Intelligence*, pages 47–52. AAAI, 2005.
2. Carlos Eisenberg. *Distributed Constraint Satisfaction For Coordinating And Integrating A Large-Scale, Heterogeneous Enterprise*. PhD thesis, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), September 2003.
3. Jeremy Frank. Learning short-term weights for GSAT. In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 384–391, San Francisco, August 1997. Morgan Kaufmann.
4. Katsutoshi Hirayama and Makoto Yokoo. Local search for distributed SAT with complex local problems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, AAMAS 2002, pages 1199 – 1206, New York, NY, USA, 2002. ACM Press.
5. Katsutoshi Hirayama and Makoto Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1–2):89–115, January 2005.
6. Katsutoshi Hirayama, Makoto Yokoo, and Kaita Sycara. The phase transition in distributed constraint satisfaction problems: firstresults. In *Proceedings of the International Workshop on Distributed Constraint Satisfaction*, 2000.
7. Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In P. Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP02)*, volume 2470 of *LNCS*, pages 233–248, London, UK, September 2002. Springer-Verlag.
8. Arnold Maestre and Christian Bessiere. Improving asychronous backtracking for dealing with complex local problems. In Ramon Lpez de Mntaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004)*, pages 206–210. IOS Press, August 2004.

9. Bart Selman and Henry A. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Principles on Artificial Intelligent (IJCAI'93)*, pages 290–294, August 1993.
10. Lars Wittenburg. Distributed constraint solving and optimizing for micro-electro-mechanical systems. Master's thesis, Technical University of Berlin, December 2002.
11. Xiaolong Jin Yi Tang, Jiming Liu. Adaptive compromises in distributed problem solving. In *Proceedings of the 4th International Conference on Intelligent Data Engineeringand Automated Learning IDEAL 2003*, March 2003.
12. Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *12th International Conference on Distributed Computing Systems (ICDCS-92)*, pages 614–621, 1992.
13. Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 401–408. MIT Press, 1996.
14. Makoto Yokoo and Katsutoshi Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS '98: Proceedings of the 3rd International Conference on Multi Agent Systems*, pages 372–379, Washington, DC, USA, July 1998. IEEE Computer Society.