Solving Coarse-grained DisCSPs with Multi-DisPeL and DisBO-wd

Muhammed Basharu
4C, University College
Cork, Ireland
mb@4c.ucc.ie

Inés Arana
School of Computing, RGU
Aberdeen AB25 1HG, UK
ia@comp.rgu.ac.uk

Hatem Ahriz
School of Computing, RGU
Aberdeen AB25 1HG, UK
ha@comp.rgu.ac.uk

Abstract

We present Multi-DisPel, a penalty-based local search distributed algorithm which is able to solve coarse-grained Distributed Constraint Satisfaction Problems (DisCSPs) efficiently. Multi-DisPeL uses penalties on values in order to escape local optima during problem solving rather than the popular weights on constraints. We also introduce DisBO-wd, a stochastic algorithm based on DisBO (Distributed Breakout) which includes a weight decay mechanism. We compare Multi-DisPeL and DisBO-wd with other algorithms and show, empirically, that they are more efficient and at least as effective as state of the art algorithms in some problem classes.

1. Introduction and background

Some problems are naturally formalised as a set of interconnected sub-problems where each sub-problem is a Constraint Satisfaction Problem (CSP) comprising a set of variables, a set of domains (one per variable) and a set of constraints between those variables. In addition, there is a set of constraints between some variables in the local CSP and variables in other sub-problems. In these coarse-grained DisCSPs, each agent is responsible for a local CSP, i.e. represents a sub-problem. Thus, besides the *inter-agent constraints* between variables held by different agents in the DisCSP there are *intra-agent constraints* between the local variables within an agent. Agents have, therefore, more problem information available and, consequently, they can carry out more (local) computation in the search for a solution without incurring communication costs. Despite the additional information available to agents when solving coarse grained DisCSPs, these can prove to be a real test for collaborative problem solving because agents have to find a balance between the emphasis they place on resolving the internal and the external constraints in order to ensure their collective ability to solve problems.

Yokoo and Hirayama [15] state that the amount of local computation required by complex agents can vary along two extremes. At one extreme, each sub-problem becomes a variable and each agent does all its local computation beforehand by finding all possible solutions for its sub-problem which are then used as the "domain values" of the variable. This approach is unsuitable for problems where local sub-problems are large and complex, since it may be impossible to find all local solutions beforehand. At the other extreme, each variable in a sub-problem acts as a virtual agent. This is wasteful because the cost of local computation is significantly lower than that of communications between virtual agents. Also, not all the information which an agent could consider during problem solving is available to it. The solution appears to lie somewhere between these two extremes.

In distributed backtracking algorithms for DisCSPs with complex agents, most prominently Asynchronous Backtracking [10] and Asynchronous Weak Commitment Search (Multi-AWCS) [15], variables are treated as virtual agents. Thus, these algorithms use a single strategy to deal with both inter-agent and intra-agent constraints, and deadlocks are still detected (and no-goods generated) from each variable rather than from entire sub-problems. The amount of local computation within agents is significant since each agent typically tries, 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 a revision of earlier choices by other agents.

Multi-DB [6] attaches weights to constraints in order to modify the cost landscape whenever quasi-local-optima are encountered. Agents act concurrently - i.e. they send value updates in the *ok?* cycle and find and coordinate concurrent improvements in the *improve* cycle. In addition, agents run a local search algorithm for a fixed number of steps to identify a set of local changes that reduces the cost of the current solution. These changes are exchanged with neighbouring agents, and the best changes are accepted. Simultaneous value changes of two or more co-constrained variables are permitted where such changes do not increase the cost of the solution.

Multi-AWCS combines backtracking and iterative improvement search and escapes deadlocks through the use of variable re-ordering and storage of explicit no-goods. While the algorithm has been shown to outperform other distributed backtracking algorithms [15], it has the drawback of possibly requiring an exponential amount of memory to store no-goods.

This paper presents two efficient and effective local search distributed constraint satisfaction algorithms for solving coarse-grained DisCSPs. The first algorithm, Multi-DisPeL, uses a penalty-based strategy to escape local optima. The second, DisBO-wd, uses a constraint-based strategy with weight-decay to avoid deadlocks.

The remainder of this paper is structured as follows. First, Multi-DisPeL is introduced in section 2. Next, DisBO-wd is explained in section 3. Finally, section 4, presents the results of empirical evaluations of Multi-DisPeL and DisBO-wd along with comparisons with other similar algorithms.

2. Multi-DisPeL

DisPeL [1] is an iterative improvement distributed constraint satisfaction algorithm where each agent controls just one variable and the objective is to find the first solution that satisfies all constraints. At each cycle, agents take turns to improve a random initialisation in a fixed order determined by the Distributed Agent Ordering scheme [5]. Thus, at initialisation, each agent locates its position in the ordering by locally partitioning its neighbours into parents (γ^+) and children (γ^-) using their lexicographic tags (or IDs).

In order to resolve deadlocks (quasi-local-optima where an agent's *view* remains unchaged for 2 iterations), DisPeL applies penalties to variable values in a 2-phased strategy : (i) In phase one, the solution is perturbed with a *temporary penalty* in order to encourage agents to try other values, therefore exploring other areas of the search space and; (ii) If phase one fails to resolve a deadlock (i.e. a deadlock is revisited), agents try to learn about and avoid the value combinations that caused the deadlock by increasing the *incremental penalties* attached to the culprit values. Whenever an agent detects a deadlock and has to use a penalty, it imposes the penalty on its current assignment and asks its neighbours to impose the same penalty on their current assignments as well. In addition, a no-good store is used to keep track of the deadlocks encountered. The cost function for each agent is as follows:

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

where: d_i is the ith value in the variable's domain; $v(d_i)$ is the number of constraints violated if d_i is selected; $p(d_i)$ is the incremental penalty attached to d_i and t is the temporary penalty ($t = 3$ in most experiments [1]).

Stoch-DisPeL [2] is a stochastic variation of DisPeL where if an agent is at a quasi-local-optimum it randomly decides to either impose a temporary penalty (with probability p) or to increase the incremental penalty (with probability $1 - p$). It, therefore, does not need to keep track of previous deadlocks and, consequently, does not require a no-goods store. For most problem types p = 0.3 gives the best results.

DisPeL and Stoch-DisPeL have been shown to be effective and efficient when compared to state-of-the-art local search algorithms [1, 2].

A contribution of this paper is Multi-DisPeL (see Algorithm 1) - a penalty-based algorithm for solving coarse-grained problems. In Multi-DisPeL both variables and agents have unique IDs and agents know the owners of variables constrained with their local variables. Hence, agent's neighbours are those agents whose variables share at least one constraint with the variables belonging to the agent. At initialisation, agents create an ordering using the Distributed Agent Ordering heuristic with their IDs, as done in DisPeL. Agents will, therefore, treat all variables belonging to higher priority agents as higher priority variables and variables belonging to lower priority agents as lower priority variables. Each agent communicates with both sets of neighbours and takes its turn to improve the solution after receiving updates from all higher priority neighbours. During the initialisation process, agents also initialise their local variables with random instantiations and exchange these assignments with their neighbours. Multi-DisPeL is outlined in detail in Algorithms 1 to 6.

In each cycle, each agent uses a modified steepest descent local search (Algorithm 3) in order to minimise the total number of constraints violated by its local variables as follows: (i) Find all improvements to the solution for

Algorithm 1 Multi-DisPeL: Main loop

```
initialise
$ordering \leftarrow empty$
for $i = 0$ to $OwnVars.size$ do
  $ordering \leftarrow ordering \wedge var_i$
end for
loop
  $messgs \leftarrow accept()$
  while active do
    for $i = 0$ to $num(messgs)$ do
      processMessage($messg_i$)
    end for
    for $i = 0$ to $Vars.size$ do
      if $var_i$ is consistent or cost function for $var_i$ is distorted then
        reset $var_i.incrementalPenalties$
      end if
      if $var_i.penaltStat \neq null$ then
        implement penalty on $var_i$
      end if
    end for
    improveSolution()
    send var. values & penalty requests to all neighbours
  end while
end loop
```

each variable; (ii) Implement simultaneously all the best improvements which are compatible, breaking ties in favour of variables with the largest number of constraints or the highest lexicographic IDs.

When penalties are used, variables are treated as virtual agents, i.e. penalties are "sent" from individual variables involved in deadlocks to variables sharing constraints with them. Like in Stoch-DisPeL, agents choose randomly between using a temporary penalty or an incremental one. However, agents do not have to detect a quasi-local-optimum in order to apply penalties. Rather, penalties are used as soon as the underlying search, which agents use individually, is stuck. Since agents do not distinguish between internal and external constraints, the steepest descent search could go on indefinitely i.e. an agent keeps trying to find a consistent local solution when values of external variables prevent it from doing so. To avoid this problem, when the steepest descent is stuck agents do not impose any "new" penalties on variables whose assignments have changed or have been penalised in the current (Algorithm 5, lines 8-10).

The steepest descent search terminates, in each iteration, when the local solution is consistent, no further improvements can be found, or agents cannot impose any new penalties on the local variables. When this happens, agents send the new variables' assignments to all affected neighbours, as well as any requests to impose penalties.

Algorithm 2 procedure processMessage($messg$)

```
1: update $AgentView$ with $messg.variable, messg.value$
2: if $messg.penaltyRequest = null$ then
3:   return
4: end if
5: for each $var_i$ constrained with $messg.variable$ do
6:   if $messg.penaltyRequest = increaseIncPenalty$ then
7:     $var_i.penaltStat \leftarrow increaseIncPenalty$
8:   else
9:     if $v_i.penaltStat \neq increaseIncPenalty$ then
10:      $var_i.penaltStat \leftarrow imposeTempPenalty$
11:    end if
12:  end if
13: end for
```

Algorithm 3 procedure improveSolution()

```
1: for $i = 0$ to $Vars.size$ do
2:   $x_i.moved \leftarrow FALSE$
3: end for
4: while true do
5:   $improvements \leftarrow$ getImprovements()
6:   if $improvements = \emptyset$ then
7:     $penaltyImposed \leftarrow$ imposePenalties()
8:     if $\neg penaltiesUsed$ then
9:       break
10:    end if
11:  else
12:    for $i = 0$ to $improvements.size$ do
13:      $x \leftarrow improvements_i.var$
14:      update $< x, improvements_i.value >$
15:      $x.moved \leftarrow TRUE$
16:    end for
17:  end if
18: end while
```

Like Stoch-DisPeL, Multi-DisPeL is sound, incomplete and terminates only if a solution is found. Multi-DisPeL's space requirements are minimal since the only additional information agents store are the penalty vectors for each variable. Hence, the space complexity increases only linearly with the problem size.

Multi-DisPeL was compared, empirically, to Stoch-DisPeL. Table 1 shows results of experiments on critically difficult, solvable, random distributed graph colouring problems with $k = 3$, $degree = 4.7$ (for a description of how coarse-grained problems were generated see section 4). The results show a substantial reduction in costs with Multi-DisPeL, confirming that problem solving is quicker when agents do additional computation when dealing with coarse-

Algorithm 4 getImprovements()

```
1: $impSet \leftarrow \emptyset$
2: for $i = 0$ to $Vars.size$ do
3:     get $v \in d_i$ with the minimum $h(x_i)$
4:     $\delta \leftarrow h(x_i.currentValue) - h(x_i.v)$
5:     if $\delta > 0$ then
6:         $impSet \leftarrow impSet \cup improvement(x_i, v, \delta)$
7:     end if
8: end for
   $bestImps \leftarrow \emptyset$
9: for all $(improvement, x_i, v_i, \delta_i) \in impSet$ do
10:    $remove \leftarrow FALSE$
11:    for all $(improvement, x_j, v_j, \delta_j) \in impSet$ do
12:        if $\neg isNeighbour(x_i, x_j)$ then continue
13:        if $\delta_i < \delta_j$ then $remove \leftarrow TRUE$
14:        if $\delta_i = \delta_j$ then
15:            if $(x_i.Constr < x_j.Constr) \vee (x_i.id > x_j.id)$ then
16:                $remove \leftarrow TRUE$
17:            end if
18:        end if
19:    end for
20:    if $\neg remove$ then
21:        $bestImps \leftarrow bestImps \cup improvement(x_i, v_i, \delta_i)$
22:    end if
23: end for
24: return $bestImps$
```

Algorithm 5 imposePenalties()

```
1: $penaltyImposed \leftarrow FALSE$
2: for $i = 0$ to $Vars.size$ do
3:     if $isConsistent(x_i) \vee$ cost function of $x_i$ is distorted then
4:         $x_i.resetIncrementalPenalties$
5:     end if
6: end for
7: for $i = 0$ to $Vars.size$ do
8:     if $x_i.moved \vee isConsistent(x_i) \vee (x_i.penaltStat \neq null)$ then
9:         continue
10:    end if
11:    $r \leftarrow$ random value in [0..1]
12:    if $r < p$ then
13:        $x_i.penaltStat \leftarrow sentAddTempPenalty$
14:    else
15:        $x_i.penaltStat \leftarrow sentIncreaseIncPenalty$
16:    end if
17:    $penaltyImposed \leftarrow TRUE$
18:    implLocPenalts(getRecipients$(x_i)$, $x_i.penaltStat$)
19: end for
20: return $penaltyImposed$
```

grained DisCSPs, as opposed to treating each variable as a virtual agent. This is unsurprising given that in Multi-DisPeL agents have more information available to them.

3. DisBO-wd

Multi-DB [6, 7] is a Distributed Constraint Satisfaction algorithm for coarse-grained DisCSPs, which is particularly effective at solving distributed SAT problems. DisBO [3] is an extension which increases weights only at real local optima. Thus, DisBO has an additional third cycle for global state detection. At each improvement phase the agents select values for their intra-agent variables (those only involved in intra-agent constraints) that minimise the weighted constraint violations until no further improvements are possible. The inter-agent variables, i.e. those involved in at least one inter-agent constraints, on the other hand, are treated like virtual agents and a coordination heuristic is used to prevent any two inter-agent 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.

We have produced DisBO-wd, an algorithm based on DisBO where the weight update scheme is replaced with a weight decay scheme similar to Frank's [4]. Instead of modifying weights only when a search is stuck at local optima, weights on violated constraints are continuously updated after each move, so DisBO's global state detection phase is not required. At the same time, weights are decayed at a fixed rate ($dr < 1$) during the updates allowing focus on recent increments. Thus, the weight on a violated constraint at time t is $W_{i,t} = (dr * W_{i,t-1}) + lr$ where lr is the learning rate ($lr > 0$). In addition, weights on satisfied constraints are continuously decayed ($W_{i,t} = max((dr * W_{i,t-1}), 1)$ and the coordination heuristic is replaced with the random break as in [13]. Other alternatives for improving DisBO were also considered, notably probabilistic weight resets and probabilistic weight smoothing [9], which both outperformed DisBO but were not as strong as the weight decay strategy.

We run a set of experiments which evaluated the performance of DisBO and DisBO-wd. Figure 1 summarises empirical results from experiments on critically difficult distributed graph colouring problems ($k = 3, degree = 4.7$) which show that DisBO-wd solved more problems than DisBO (Figure 1), especially on the larger problems. Furthermore, DisBO-wd required substantially fewer cycles on average to solve the problems.

Algorithm 6 implLocPenalts($recipientList, penaltySent$)

```
1:  for each $x_i \in (recipientList.size \cap Vars)$ do
2:    if $penaltySent = addTempPenalty$ then
3:      if $x_i.penaltStat = null$ then
4:        $x_i.penaltStat \leftarrow imposeTempPenalty$
5:        impose temporary penalty on $x_i.currentValue$
6:      end if
7:    else
8:      if $x_i.penaltStat = null \vee x_i.penaltStat = imposeTempPenalty$ then
9:        $x_i.penaltStat \leftarrow increaseIncPenalty$
10:       increase incremental penalty on $x_i.currentValue$
11:     end if
12:   end if
13: end for
```

4. Evaluation of Multi-DisPeL and DisBO-wd

We evaluated Multi-DisPeL and DisBO-wd on random DisCSPs, distributed SAT problems and distributed graph colouring problems. We studied the relationship between search costs and the problem size, as well as the influence of agent size (i.e. the number of variables each agent controls) on its performance. In the experiments, each algorithm was limited to a maximum of $100n$ iterations in each attempt where n is the number of variables. However, a maximum of $200n$ iterations were used for DisBO-wd, to account for its two cycles (i.e. *improve* and *ok?*) and as such give it the same number of opportunities to change variable assignments as the other algorithms. For each problem, we generated 100 instances and recorded the percentage of problems solved within the maximum number of iterations, the average and median number of iterations required.

Since Multi-DisPeL gives a better performance than Stoch-DisPeL (see section 2) and the latter has been shown to be more effective and efficient than other non-coarse-grained DisCSP algorithms such as DBA [14] and DSA [16], it can be concluded that Multi-DisPeL is more efficient and effective than these algorithms.

Multi-DisPeL and DisBO-wd's performance was compared to that of Multi-AWCS [15]. Since Multi-DB works particularly well on distributed SAT problems and it has been shown to outperform Multi-AWCS in that domain we use Multi-DB in the experiments on distributed SAT problems. Note that given its completeness, Multi-AWCS is guaranteed to solve all problems used since they all have solutions but we are evaluating its performance in bounded time.

The problems used in the experiments were partitioned into inter-connected sub-problems for each agent using a simple partitioning algorithm. For each agent a_i: (i) select a non-allocated variable x_i and allocate it to a_i; (ii) repeatedly select a non-allocated variable connected to a variable in a_i and allocate it to a_i until the right number of variables is given to a_i; with a very small probability, step (ii) will choose a non-allocated variable possibly not connected to any variable in a_i.

	num. vars.	number agents	% solved	avrg. cost	median cost
Stoch-DisPeL	100	-	100	236.5	111
	150	-	100	686.4	300
	200	-	99	1878.5	890
	250	-	98	2201.2	1277
Multi-DisPeL	100	2	100	84.5	44
		4	100	102.6	43
		5	100	105.3	58
		10	100	112.1	55
	150	3	100	300.7	110
		5	99	271.3	121
		10	100	291.2	148
		15	100	351.1	135
	200	4	100	804.4	329
		5	100	897.3	324
		10	100	1135.4	373
	250	5	99	1242.6	417
		10	100	1660.5	529
		25	97	1785.7	668

Table 1. Performance of Multi-DisPeL and Stoch-DisPeL on distributed graph colouring problems.

Random DisCSPs were used to evaluate the algorithms on three groups of 100 problems with varying sizes ($< n, d = 10, p1 \approx 0.1, p2 = 0.5 >$) - see results in Table 2. Note that for Multi-AWCS results were only obtained in the runs with 50 variables due to the exponential amount of memory it requires to store no-goods [12][1]. Multi-DisPeL generally gives a better performance and lower search costs than Multi-AWCS and DisBO-wd. The average and median search costs for Multi-DisPeL show a steady increase as the number of agents increases whereas DisBO-wd's performance degrades considerably on the largest problems.

Distributed SAT problems (satisfiable 3-SAT) from the SATLib dataset [8] made up of formulae with 100, and 150 literals were used to compare the algorithms. For Multi-DB and Multi-AWCS, we used results on experiments with the same instances from [6] - note, however, that variables are randomly distributed amongst agents in [6], so the distribution may not be identical to ours. For Multi-DisPeL, the temporary penalty (t) was fixed to 2 and the probability of using the temporary penalty (p) was set to 0.5. The results of the experiments (see Table 2) show that, in terms of % of problems solved, DisBO-wd does best followed by Multi-DB which performed slightly worse. Multi-DisPeL,

[1]We used Java on a 3Ghz Pentium PC with 1GB of RAM.

Random DiscSPs					
algorithm	n	agents	%	avrg. c.	medn. c.
Multi-DisPeL	50	5	99	**307**	**124**
		10	**99**	**309**	**146**
	100	5	**97**	**856**	**276**
		10	**94**	**928**	**449**
	200	5	**95**	**1382**	**534**
		10	**95**	**2190**	**727**
Multi-AWCS	50	5	**100**	738	288
		10	98	995	527
	≥ 100		out	of	memory
DisBO-wd	50	5	94	1927	1336
		10	**99**	1855	1104
	100	5	83	4996	2922
		10	88	4695	3065
	200	5	62	13454	8060
		10	65	16832	14432
SAT Problems					
algorithm	lits	agents	%	avrg. c.	medn. c.
Multi-DisPeL	100	5	98.1	**487**	**136**
		10	98.7	**593**	**154**
	150	5	90	**829**	**268**
		10	93	**1214**	**292**
Multi-DB	100	5	**100**	1640	570
		10	99.6	3230	1150
	150	5	**100**	3230	1200
		10	96	9030	2090
Multi-AWCS	100	5	97.6	6100	1730
		10	96.8	7630	2270
	150	5	67	37100	26100
		10	61	39400	36000
DisBO-wd	100	5	**100**	984	490
		10	**99.9**	1003	516
	150	5	99	2186	910
		10	**99**	2054	1012
Graph colouring problems					
algorithm	n	agents	%	avrg. c.	medn. c.
Multi-DisPeL	100	5	**100**	**105**	**58**
		10	**100**	**112**	**55**
	150	5	99	**271**	**121**
		10	**100**	**291**	**148**
	200	5	**100**	897	**324**
		10	**100**	1135	**373**
Multi-AWCS	100	5	**100**	123	92
		10	**100**	163	129
	150	5	**100**	288	198
		10	**100**	341	277
	200	5	**100**	**556**	431
		10	**100**	**704**	527
DisBO-wd	100	5	**100**	1084	691
		10	**100**	1020	817
	150	5	97	4376	2274
		10	98	4977	2482
	200	5	82	11727	6686
		10	83	10262	5956
Multi-DisPeL*	100	7.2†	**100**	126	**61**
	150	9.6†	99	**304**	**129**
	200	11.9†	**100**	1056	**348**
Multi-AWCS*	100	7.2†	**100**	**121**	98
	150	9.6†	**100**	**304**	263
	200	11.9†	**100**	**672**	558
DisBO-wd*	100	7.2†	**100**	1100	568
	150	9.6†	**100**	3872	2401
	200	11.9†	89	11432	7414

Table 2. Algorithms' performance

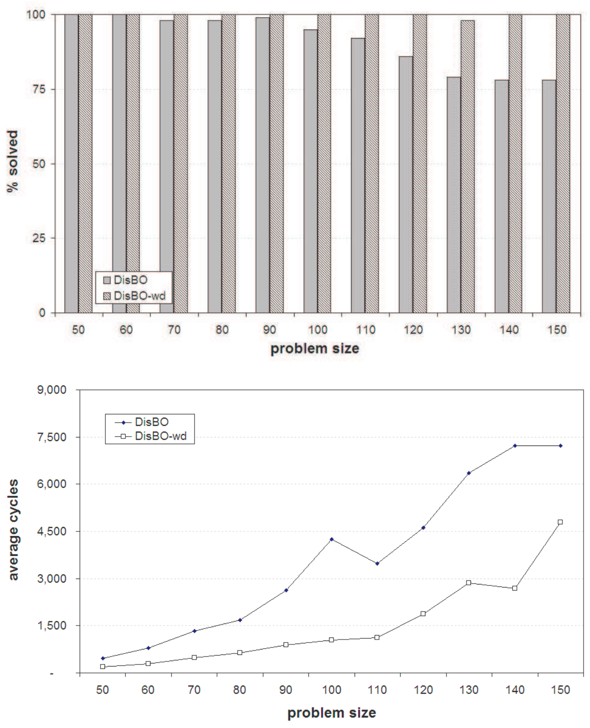

Figure 1. Comparison of success rates and average cycles required by DisBO and DisBO-wd on random distributed graph colouring problems of various sizes. Each point represents attempts on 100 problems.

does substantially better than Multi-AWCS, although it is not quite as good as the other two algorithms. Regarding cost, Multi-DisPeL does remarkably well compared to the other algorithms, with substantially lower average and median search costs. Also, DisBO-wd search costs were significantly smaller than those of Multi-DB and Multi-AWCS.

Distributed graph colouring problems were used to compare the algorithms and to study the relationship between search costs and problem size (see Table 2). 100 instances of critically difficult, solvable, random distributed graph colouring problems ($k = 3, degree = 4.7$) were generated for each problem size used. In the table, * indicates an uneven distribution of variables per agent and † indicates the average number agents per problem.

With an even distribution of variables to agents, Multi-DisPeL solved slightly less problems than Multi-AWCS but it incurred lower costs. DisBO-wd solved fewer problems and its search costs were considerably higher. When the distribution of variables to agents is uneven, Multi-AWCS solves slightly more problems than Multi-DisPeL and it has lower average but higher median costs. DisBO-wd performed substantially worse than the other two algorithms.

The results of the experiment show that Multi-DisPeL is able to achieve similar levels of performance to Multi-AWCS without the additional overhead of creating new constraints (in form of no-goods) and not breaching privacy by connecting variables that were not previously linked in the original specification of the problem being solved.

5. Discussion and conclusions

We have presented two successful algorithms for solving coarse-grained DisCSPs: (i) Multi-DisPeL - an algorithm where each agent runs a steepest descent algorithm locally and uses (temporary and incremental) penalties on individual domain values to escape from local optima and; (ii) DisBO-wd, an effective algorithm based on DisBO which incorporates weight decay and randomness and eliminates the need for a global state detection phase.

Compared to Multi-AWCS, Multi-DisPeL performed better and more efficiently in two problem classes (random DisCSPs and distributed SAT problems). Multi-DisPeL solved slightly less problems in the experiments with distributed graph colouring problems, but it had lower median and average search costs. Note that, in addition, Multi-DisPeL has the following two advantages: (i) a much lower space complexity than Multi-AWCS since it does not create any new constraints (in the form of no-goods) and; (ii) no new links are created between unconnected agents, so privacy is preserved.

Multi-DisPeL solved more problems than DisBO-wd and it required fewer iterations for two problem classes (distributed graph colouring and random DisCSPs), but DisBO-wd was better at solving SAT problems where it was the most competitive algorithm. DisBO-wd was also competitive on SAT problems compared to Multi-DB because in DisBO-wd weights are not allowed to grow unbounded so its costs are lower. Multi-DB was better than Multi-DisPeL and Multi-AWCS for SAT problems although both its average and median costs were substantially greater than Multi-DisPeL's.

Search efficiency in Multi-DisPeL could be further improved by using pre-processing techniques locally on agents. In addition, agents are not necessarily restricted to using a steepest descent search heuristic to improve their local sub-problem resolution. Other heuristics could be used locally (e.g. Novelty[11]) as long as penalties can be directly incorporated in the cost functions of such methods and agents can determine when to implement new penalties.

In summary, we have presented two algorithms for solving coarse-grained DisCSPs, i.e. Multi-DisPeL and DisBO-wd, which are effective and efficient when compared to other well-known algorithms for some problem classes.

References

[1] M. Basharu, I. Arana, and H. 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] M. Basharu, I. Arana, and H. Ahriz. Stoch-DisPeL: Exploiting randomisation in DisPeL. In *Proceedings of DCR 06 - the Seventh Intenational Workshop on Distributed Constraint Reasoning*, pages 117–131, 2006.

[3] C. 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.

[4] J. Frank. Learning short-term weights for GSAT. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 384–391, San Francisco, August 1997. Morgan Kaufmann.

[5] Y. Hamadi, C. Bessière, and J. Quinqueton. Backtracking in distributed constraint networks. In H. Prade, editor, *13th European Conference on Artificial Intelligence ECAI 98*, pages 219–223, Chichester, August 1998. John Wiley and Sons.

[6] K. Hirayama and M. 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.

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

[8] H. H. Hoos and T. Stutzle. Satlib: An online resource for research on SAT. In I. P. Gent, H. van Maaren, and T. Walsh, editors, *Third Workshop on the Satisfiability Problem (SAT 2000)*, pages 283–292. IOS Press, 2000.

[9] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In P. V. 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.

[10] A. Maestre and C. Bessiere. Improving asychronous backtracking for dealing with complex local problems. In R. L. de Mntaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004)*, pages 206–210. IOS Press, August 2004.

[11] D. A. McAllester, B. Selman, and H. A. Kautz. Evidence for invariants in local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97)*, pages 321–326. AAAI Press / The MIT Press, 1997.

[12] A. Meisels and O. Lavee. Using additional information in DisCSPs search. In P. J. Modi, editor, *Proceedings of the 5th International Workshop on Distributed Constraint Reasoning*, September 2004.

[13] L. Wittenburg. Distributed constraint solving and optimizing for micro-electro-mechanical systems. Master's thesis, Technical University of Berlin, December 2002.

[14] M. Yokoo and K. 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.

[15] M. Yokoo and K. 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.

[16] W. Zhang, G. Wang, and L. Wittenburg. Distributed stochastic search for constraint satisfaction and optimization:parallelism, phase transitions and performance. In *Proc. AAAI Workshop on Probabilistic Approaches in Search*, pages 53–59, 2002.