Escaping Local Optima: Constraint Weights vs. Value Penalties

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. Constraint Satisfaction Problems can be solved using either iterative improvement or constructive search approaches. Iterative improvement techniques converge quicker than the constructive search techniques on large problems, but they have a propensity to converge to local optima. Therefore, a key research topic on iterative improvement search is the development of effective techniques for escaping local optima, most of which are based on increasing the weights attached to violated constraints. An alternative approach is to attach penalties to the individual variable values participating in a constraint violation. We compare both approaches and show that the penalty-based technique has a more dramatic effect on the cost landscape, leading to a higher ability to escape local optima.
We present an improved version of an existing penalty-based algorithm where penalty resets are driven by the amount of distortion to the cost landscape caused by penalties. We compare this algorithm with an algorithm based on constraint weights and justify the difference in their performance.

1 Introduction

There are two main categories of search algorithms in constraint satisfaction: constructive and iterative improvement/local search. While the former algorithms are complete, the latter offer the advantage of quicker convergence, albeit often to local optima. Techniques for escaping local optima can be categorised as: (i) Strategies based on the introduction of some non-improving decisions in order to attempt to move the search away to other regions of the search space where the algorithm can resume the search for a solution, e.g. a random move with a small probability; (ii) Strategies which try to determine the sources of the deadlocks (local optima) and seek moves that directly attempt to resolve them, e.g. the breakout algorithm (BA) [8]. In BA, the shape of the objective/cost landscape is modified by increasing the weight of the constraints which are not satisfied at a local optimum. The breakout approach has been shown to be very effective for solving several types of problems and a number of algorithms have been proposed which use this technique, e.g PAWS [9].

More recently, Basharu et al. [1] proposed a technique which associates penalties to domain values, rather than to constraints albeit for solving Distributed CSPs. They

presented the DisPeL algorithm and included a comparison between DisPeL and the Distributed Breakout (DBA) [12] which showed that the former outperforms the latter in distributed graph colouring and random distributed constraint satisfaction problems.

In this paper, we analyse the weight-based and penalty-based techniques for escaping local optima (weights on constraints vs. penalties on domain values) and justify the difference in their performance. We also propose a modification to the penalty resetting strategy used in DisPeL which improves its overall performance.

The remainder of this paper is structured as follows. Section 2 gives some definitions and background knowledge. The two landscape modification techniques, i.e. constraint weights and value penalties are explained in Sections 3 and 4 respectively. We highlight a weakness of the weigthts-based approach and illustrate, through and example, why the penalty-based technique does not suffer from this disadvantage. We then propose a new penalty resetting strategy for the penalty-based algorithm DisPeL and show, empirically, its effectiveness (Section 5). Finally, Section 6 compares an algorithm based on constraint weights (DBA) with an algorithm based on value penalties (DisPeL) and justifies the latter's advantage over the former.

2 Background

A Constraint Satisfaction Problem (CSP) consists of a tuple $< X, D, C >$ where $X = \{x_1, ..., x_n\}$ is a set of variables; $D = \{d_1, ..., d_n\}$ is a set of domains, one for each variable in X and; C is a set of constraints which restrict the values that variables can take simultaneously. A solution to a CSP is an assignment for each variable in X which satisfies all the constraints in C.

A Distributed Constraint Satisfaction Problem (DisCSP) is a CSP where the problem is distributed over a number of agents $A = \{A_1, ..., A_k\}$. Each agent A_i represents a set X_i of variables it controls such that $\forall i \neq j, X_i \bigcap X_j = \phi$ and $\bigcup_{i=1..k} X_i = X$. It is often the case that an agent holds one variable only, i.e. $\forall i \in [1..k] |X_i| = 1$. Agents communicate with other agents they know of by sending messages. Without loss of generality, we assume that messages arrive to their destination in finite time and that messages are received in the order in which they were sent. We also assume that each agent is responsible for one variable only.

DisCSP are particularly challenging due to *privacy* and *partial knowledge* of a problem [11]. Thus, agents only know the domains and the constraints for the variables they represent. An agent may only reveal to other agents the current assignments to its variables. These features distinguish distributed constraint satisfaction from parallel or distributed approaches for solving traditional CSPs. DisCSPs are solved by a collaborative process, where each agent tries to find assignments for its variables that satisfy all associated constraints.

3 Modifying the cost landscape with constraint weights

Morris [8] introduced a strategy for dealing with local optima in boolean satisfiability (SAT) problem resolution by modifying the shape of the cost landscape. His technique enables the search to focus on resolving clauses (constraints) that are repeatedly unsatisfied and regularly associated with local optima by attaching weights to these constraints. The cost function of the problem to be solved is as follows:

$$h = \sum_{i=1}^{n} cw_i * viol(c_i) \tag{1}$$

where:
c_i is the *i*th constraint
cw_i is the weight of the *i*th constraint
$viol(c_i)$ is 0 if the constraint is satisfied, 1 otherwise.

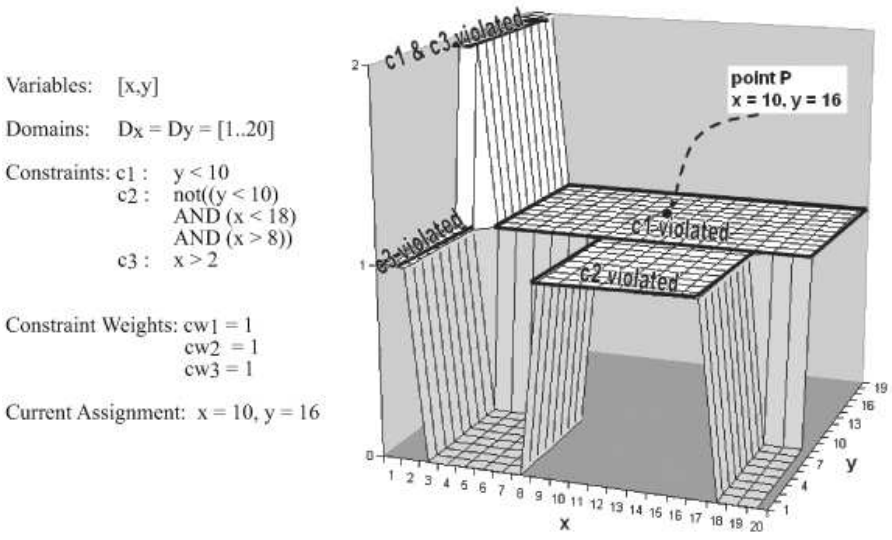

Fig. 1. A CSP and its cost landscape

We illustrate the use of weights on constraints using two examples. In these examples, when two or more variable values lead to the same maximal improvement, the first (lowest) value is chosen. First we look at the simple CSP and its cost landscape in Figure 1. There is a raised area at cost 1 where only c_1 is violated, another one where only c_2 is violated and a third one where only c_3 is violated. There is also a raised

area at cost 2 where both c_1 and c_3 are violated. In addition, there are two regions with solutions, i.e. all constraints are satisfied.

Initially, the problem is in state $x = 10, y = 16$ (point P in Figure 1) and constraint c_1 is violated (cost 1). No change in an individual variable improves the situation: (i) changing the value of x does not affect the truth value for constraint c_1; (ii) changing the value of y could change the truth value of c_1, but only at the expense of violating constraint c_2. As both c_1 and c_2 have the same associated weight (1) changing the value of y does not improve the overall cost. Since c_1 is violated at point P, its weight cw_1 is increased to 2, changing the shape of the cost landscape (see Figure 2). The additional weight carried by constraint c_1 has raised point P making improvements possible as follows: (i) y can change its value to 1, hence satisfying the heavy constraint c_1, but violating the light constraint c_2 (point Q in Figure 2); (ii) finally, variable x can change its value to 3 so all constraints are satisfied (point R in Figure 2). Therefore, the problem has been solved and R ($x = 3, y = 1$) is the solution found.

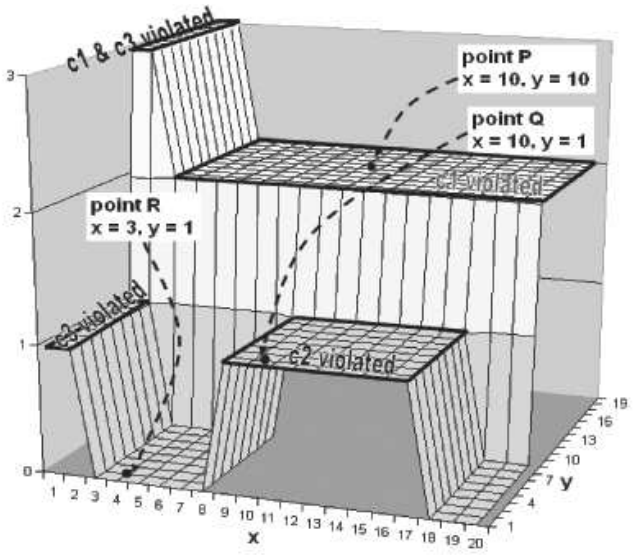

Fig. 2. Cost landscape for the CSP in Figure 1 with increased weights for c_1

It can be seen from the example that when the search is stuck at a local optimum - in this case within a plateau - the weights attached to violated constraints are increased, thus changing the shape of the cost landscape so that the search can resume downhill. However, there are problems for which this technique is ineffective because for some deadlocks it does not raise the current state compared to its surrounding areas so the search cannot find a path out of the local optimum. The example CSP in Figure 3 illustrates this problem and shows the cost landscape given the current assignment. Here, the search is in a deadlock state (point P) on a plateau where only c_1 is violated. There are two other plateaus in the landscape: the region where both constraints are

violated and the region where just c_2 is violated. There is also a small region with solutions (e.g., $x = 11, y = 1$).

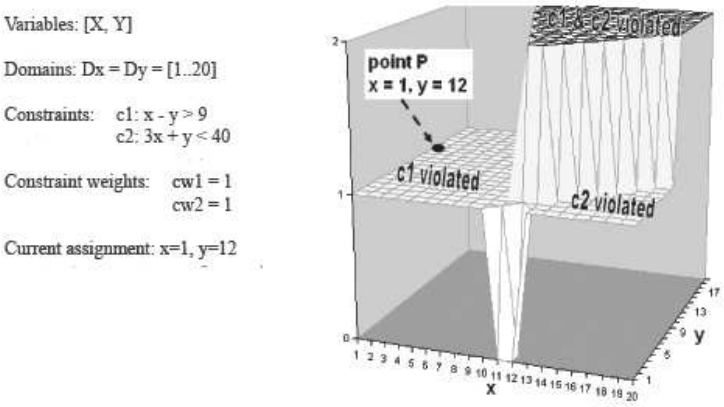

Fig. 3. A CSP and its cost landscape

An attempt to resolve the deadlock using constraint weights increases the weight cw_1 of the violated constraint c_1 to 2, producing the cost landscape in Figure 4. It can be seen from this figure that the problem solving is still in a deadlocked state, since not only the plateau containing the deadlock has been raised, but also every possible point the search could consider moving to within one step. The search is, therefore, unable to find a path out of the plateau and the deadlock remains unresolved. Increasing the weight on c_1 any further simply raises the plateau containing the deadlock and all points the search could consider moving to, but the conflict remains, albeit at a higher altitude. This problem with the given initialisation can, therefore, not be solved using the breakout technique (weights on constraints) alone.

4 Modifying cost landscapes with penalties on domain values

The DisPeL algorithm for solving Distributed CSPs [1], includes the technique of associating penalties with domain values, rather than weights on constraints, in order to escape local optima. Thus, the emphasis is on the assignments associated with constraint violations at local optima rather than on the constraints violated at that point. As a penalty is attached to each individual value in every variable's domain, the cost function to be minimised by the search for each variable is as follows:

$$h(d_i) = v(d_i) + p(d_i) \tag{2}$$

where:

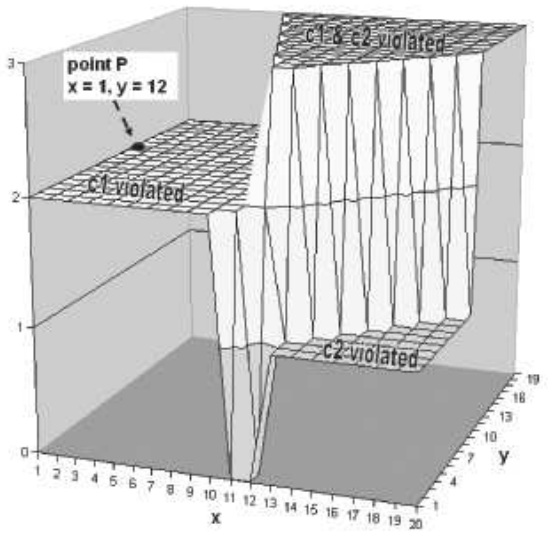

Fig. 4. Effect of constraint weight increase on the cost landscape for the CSP in Figure 3.

d_i is the *i*th 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 penalty attached to d_i (initially 0)

When the underlying search is stuck at a local optimum, penalties attached to the values currently assigned to the variables involved in violated constraints are increased, therefore contorting the cost landscape around the deadlocked region. In order to illustrate landscape modification with this technique, the two examples from section 3 (Figures 1 and 3) are used. As with the breakout algorithm, when two or more variable values lead to the same maximal improvement, the first (lowest) value is chosen.

Let's first look at the example in Figure 1. Its initial cost landscape is shown in it. Since the search is stuck at a local optimum (point P in Figure 1) the penalty on the current value of y (i.e. the variable involved in the violated constraint c_1) is increased, resulting in the cost landscape shown in Figure 5. It can be seen from this figure that a new ridge has appeared which contains the point of the deadlock P. The search can now resume downhill with $y = 1$ (point Q), followed by $x = 3$ (point R). Note that if a value of y had been chosen which prevents x from changing its value too (e.g. $y = 14$) the penalties would have been applied to this value, causing a new ridge in the cost landscape from which the search could be resumed.

Let's now look at the example in Figure 3 which could not be solved using the breakout. The initial cost landscape is also shown in this figure. Since constraint c_1 is violated and there is no possible improvement, a penalty is attached to the values of variables involved in c_1. The values $x = 1$ and $y = 12$ are therefore penalised, contorting the cost landscape (see Figure 6) with new peaks and ridges from which the search can resume as follows: (i) variable x changes its value to 2 (point Q); (ii) y

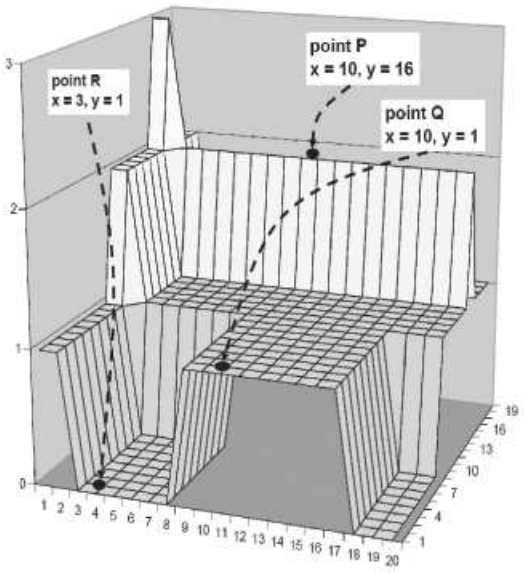

Fig. 5. Effect of penalty increases on the cost landscape for the CSP in Figure 1 i.e. increased penalty attached to $D_y(16)$ from 0 to 1.

changes its value to 1 (point R); (iii) finally, x changes its value to 11 (point S), and the problem is solved.

One contribution of this paper is the explanation of why a penalty-based approach can outperform a weight-based approach. The former technique is finer-grained and allows for more dramatic contortions in the cost landscape, causing peaks and ridges to appear, as illustrated with the example in Figure 3. Besides contorting plateaus, domain penalties may be thought of as a primitive form of learning. As penalties attached to particular values grow, the search is able to gradually *learn* of the association between the assignments and local optima. Hence, regions containing those assignments are excluded from further exploration as the search progresses.

5 Penalty resets guided by the cost landscape

Based on the technique of escaping local optima by associating penalties to domain values, DisPeL [1] uses two types of penalties (incremental and temporary) in a two phased strategy as follows:

1. In the first phase, the solution is perturbed by attaching a *temporary penalty* to the culprit variables values in an attempt to force agents to try other combinations of values, and allow exploration of other areas of the search space. This phase is used

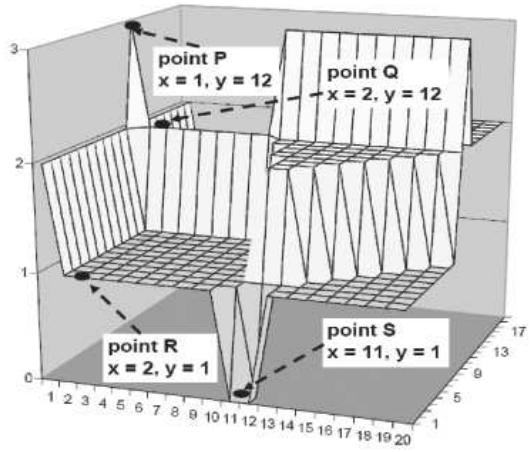

Fig. 6. Effect of penalty increases on the cost landscape for the CSP in Figure 3, i.e. increased penalties attached to $D_x(1)$ and $D_y(12)$ from 0 to 1.

when a local optimum is visited for the first time or when it hasn't been visited for a long time.

2. If the perturbation fails to resolve a deadlock, i.e. a recent 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.
3. 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.
4. A no-good store is used to keep track of deadlocks encountered, and hence, used to help agents decide what phase of the resolution process to initiate when a deadlock is encountered.

The incremental penalty is small and accumulative and its effect lasts for a number of cycles. The temporary penalty, on the other hand, is larger and it is discarded as soon as it is used, i.e. it only lasts one cycle.

The use of weights on constraints has been shown to have undesirable effects on the cost landscape [8, 10] and, therefore, there are mechanisms to undo their impact. Similarly, the DisPeL algorithm uses a periodic penalty reset every 6 cycles, thus resetting all penalties to zero. The empirical justification for resetting every 6 cycles in [1] is not entirely convincing as other values appeared to give similar results. In addition, it is unclear how this reset policy affects different types of problems.

Here, we show that the penalties on values used by DisPeL can easily dominate cost functions, since they are given equal weighing with constraint violations (see Equation 2). This can drive the search away from promising regions because, as penalties grow, the search is moved towards regions where values have the least penalties rather than the least constraint violations. Moreover, the effect that penalties have

in the cost landscape depends on the problem at hand, rather than on the number of cycles elapsed since the penalties were imposed.

A contribution of this paper is a resetting strategy based on the effect of penalties on the cost landscape, rather than on the number of cycles elapsed. Thus, the penalties of all values in a variable's domain are reset to zero, after variable assignment, in the following situations:

Consistency: When a variable has a consistent value as it is assumed that penalties become redundant when a variable has a consistent value.

Distortion: When a variable's cost function is distorted. This is detected when the following two conditions are satisfied simultaneously:

- **Condition 1:** The evaluation (h) of the current assignment is the least in a variables domain.
- **Condition 2:** There is another value in the domain which has fewer constraint violations than the current assignment.

The consistency rule simply states that if a variable's assignment is not causing problems currently, the cases when it was a problem in the past can now be ignored. This gives the algorithm room to manoeuver when the variable's assignment suddenly becomes inconsistent or has to partake in deadlock resolution with inconsistent neighbours.

The distortion rule is akin to an aspiration move (as in Tabu search [4]), where search memory is sometimes ignored. This is illustrated with the example in Figure 7.

$D_x = [w, r, b, g, k]$ $\quad v(D_x) = [3, 5, 2, 3, 6]$ $\quad p(D_x) = [2, 0, 4, 1, 0]$

therefore,

$h(D_x) = [5, 5, 6, 4, 6]$ and the current assignment with the least $h(D_x)$ is $< x = g >$.

Fig. 7. An example of a distorted cost function.

In this example, the cost function is distorted because the assignment $< x = g >$ has the least sum of constraint violations and penalties whereas the assignment $< x = b >$ violates fewer constraints and as such the cost function is being distorted by the penalties. Thus, penalties have become dominant in decision-making. Resetting penalties when this happens allows the algorithm to keep paths to solutions open.

Despite the clear advantage of penalty resets, using these too frequently could cause the search to oscillate as it continuously removes the (recently put up) barriers that prevent it from returning to previously visited (and infeasible) regions of the search space. An empirical analysis of the benefits of our new reset strategy when compared to the risks of frequent reset has been conducted. Table 1 shows the results of experiments for solving 100 randomly generated DisCSPs which clearly indicate that penalty resets are necessary. They also show the positive impact of the new penalty

reset strategy on the percentage of problems solved (all problems used were solvable), and on the average search cost (100 run/problem instance were used).

Penalties differ from constraint weights fundamentally in the way they affect cost functions. While weights can be seen as fully integrated into an underlying function, penalties are more like modifiers of an underlying function. Hence, while solvers that rely on constraint weights can successfully solve problems without limiting the growth of weights, it appears that this is not the case with the penalty based strategy. It is clear that penalties do a very good job at blocking off paths to solutions if retained for too long a period. Performance improves when penalties are discarded frequently: more problems are solved and the search costs are at least 78% lower.

Table 1. Comparative evaluation of alternative reset policies for solving 100 randomly generated CSPs $< number\ of\ variables\ n = 60, domain\ size\ d = 10, density\ p_1 = 0.1, tightness\ p_2 = 0.5 >$.

Policy	% solved	average cost
No resets	1	n/a
Reset when consistent	87	1216
Reset when distorted	95	1063
Combined reset	99	275

We have incorporated our reset policy into the DisPeL algorithm and conducted experiments using both the old and the new penalty reset policies in DisPeL in order to solve graph colouring problems with 3 colours and 100 variables for which complexity peaks are well established [2]. We have considered graphs with varying degrees (constraint densities), and for each we have generated 100 solvable instances. Figure 8 shows the median search costs using both the old and the new reset policies. It is clear that the new reset policy leads to a substantial reduction in the cost of DisPeL. In the remainder of this paper, whenever we mention DisPeL we refer to the algorithm with the new penalty reset strategy unless otherwise stated.

Similar ideas of discarding search memory (in the form of weights or penalties) have been explored in the literature. For example, periodic penalty resets were proposed in [7], while regular [3] and probabilistic [6] weight decays have been shown to improve performance of weighted local search algorithms. There, the authors also argue that weights can block paths to solutions when retained. However, there is no equivalent in the literature for resetting penalties when variables find consistent assignments. Also, using both conditions in our distortion rule for resetting penalties is new, as far as we are aware.

6 DBA vs. DisPeL

In order to evaluate the two strategies for escaping local optima we have conducted a number of experiments comparing the constraint-weights and the value-penalty ap-

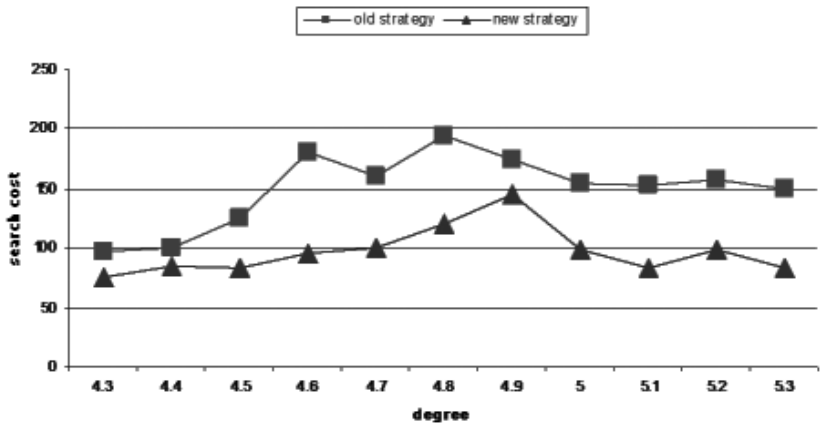

Fig. 8. Comparison of the median search cost of the old (every 6 cycles) and new (whenever the cost landscape is distorted) reset policies in solving graph colouring problems $< number\ of\ variables\ n = 100, number\ of\ colours\ k = 3, degree = 4.7 >$.

proaches. Since the original penalty-based algorithm (DisPeL) was distributed and there are successful distributed versions of the weighted constraint algorithms (e.g. DBA [12] and Multi-DB [5]) we have compared the two (weight- and penalty-based) strategies for escaping local optima by comparing DBA and DisPeL. The maximum number of iterations was set to $200n$ for DBA and $100n$ for DisPeL, thus taking into account DBA's 2-iteration cycles and making our experiments comparable to those reported in [1].We used graph colouring, random and car sequencing problems for our evaluation. Basharu et al. conducted experiments with the first two types of problems [1], albeit with their version of DisPeL (with the 6-cycle reset strategy). Their results showed that DisPeL outperformed DBA, solving substantially more problems at significantly smaller costs. The results that we have obtained show an even better performance for our version of DisPeL (with the new reset penalty) so, for reasons of space, these results are not included here.

Car sequencing is not one of the traditional test beds used for evaluating distributed algorithms, but it was used because DisPeL and DBA have not been previously compared on structured problems or problems with non-binary constraints and the car sequencing dataset [3] is one of the few publicly available problem sets where instances are entirely made up of non-binary constraints. Besides, it allows us to present results from problems which have not been randomly generated.

We used instances from the suite of feasible test problems. These are made up of 50 instances, grouped into 5 sets of 10 instances for each of the different workstation

[3] Prob001 from CSPLib at http://www.csplib.org [accessed 26 March 2007]

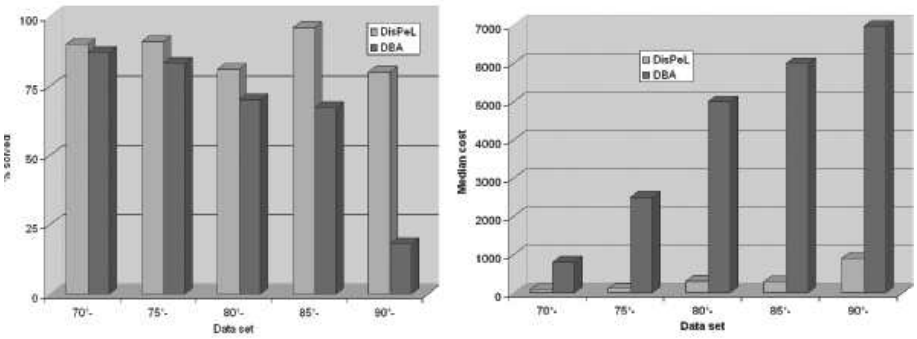

Fig. 9. Distributed car sequencing: percentage of problems solved and median costs by DisPeL and DBA.

capacity rates (or constraint tightnesses) which range from 70% to 90%. In each of these instances there are 200 cars to schedule, 5 workstations, and 17 to 30 configurations of options (or models) to be considered. To generate enough data for analysis, we made 5 attempts on each problem instance starting off with a different random initialisation in each run. The maximum number of iterations was set to 5,000 for DisPeL and 10,000 for DBA. Results of these experiments are presented in Figure 9 showing the percentage of problems solved and the median search costs for the problems solved.

The empirical results comparing DisPeL and DBA which we have obtained demonstrate the strengths of the penalty based strategy for dealing with local optima. We note that for reasons of space not all results are presented here, and remind the reader that the results that we have obtained on graph colouring and random problems show an improvement of DisPeL due to the new reset strategy which we have introduced when compared to results reported in [1]. DisPeL consistently solved more problems than DBA and it required fewer iterations. While the differences in performance of both algorithms vary from one problem class to the next, it appears that the gap widens as constraints get tighter. For example, in distributed graph colouring all constraints have a uniform tightness of 30% and DBA solves nearly as many problems as DisPeL. But as constraint tightness is increased to 50% in the experiments with random DisCSPs, DBA's performance degrades considerably. This is even more evident in the results of the experiments on the car sequencing problems which include constraints with even higher tightness.

The experiments with the car sequencing problems illustrate how DBA is adversely affected by the structure of the constraint graph. Its coordination heuristic is designed to prevent connected agents from changing values concurrently to avoid oscillation. But in the case that each variable is connected to every other variable in the problem, only one variable's value is changed in every two iterations. Therefore, search costs are inevitably high.

We explain DisPeL's performance advantage over DBA as follows:

– Landscape modification with domain penalties is more effective at dealing with local optima and it allows quicker resumption of search by the underlying algorithm, compared to modifications with constraint weights.

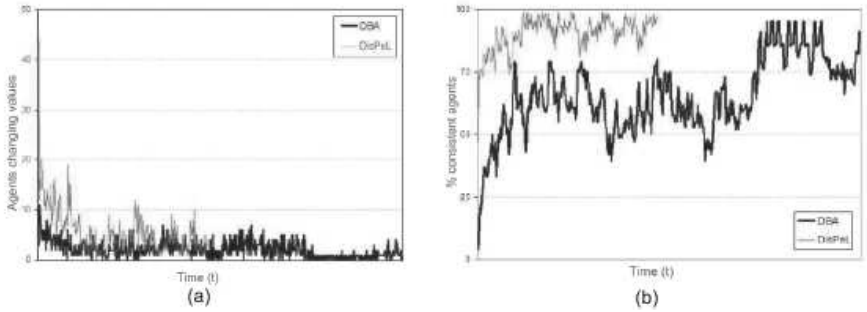

Fig. 10. Number of agents changing values and number of consistent agents in each iteration for sample runs of DBA and DisPeL.

- The new penalty reset strategy gives the algorithm opportunities to undo negative effects of penalties on the cost landscape. DBA, however, has no such opportunities and as such its performance can be severely hindered by bad decisions made early in the search or ill-advised weight increases.
- While parallel computation in DBA allows agents to reduce idle time, the coordination heuristic can slow the algorithm down considerably by inadvertently cutting down on the number of legal improvements that can take place in a single iteration (see Figure 10 (a)). We observed that there is a lot more activity in each iteration of DisPeL than DBA (in terms of agents changing values) especially in the first few iterations where high percentage of agents quickly become consistent, i.e. only 'critical' deadlocks remain unresolved. For DBA, however, few agents get to change values in each iteration therefore deadlocks tend to linger during the search. As a result, the number of consistent agents increases at a much slower pace than in DisPeL (see Figure 10 (b)).

In summary, the results of comparative evaluations of DisPeL and DBA show that on different problem types, DisPeL solved more problems than DBA and incurred lower search costs in the process. The results also suggest that DisPeL's advantage over DBA widens as constraint graphs become denser, especially since in highly connected graphs DBA's heuristics limit the number of concurrent changes per single iteration.

7 Conclusion

The contributions of this paper are threefold: a comparative analysis of the weight-based vs. the penalty based approach to escaping local optima in local search; a new reset strategy for the penalty-based approach to escaping local optima; an explanation for the difference in performance of the two landscape modification approaches to escaping local optima.

We have presented an analysis of weights on constraints (exemplified by the Breakout) and penalties on domain values (exemplified by DisPeL). We have discussed the way in which each technique modifies the cost landscape, uncovering a limitation of

the first technique in cases where the problem gets stuck in a plateau. We have also justified the suitability of the penalty-based technique for solving deadlocks of this type.

Moreover, we have introduced a novel penalty resetting strategy in DisPeL based on the level of distortion to the cost landscape produced by the penalties. We have demonstrated the effectiveness of this new policy when compared to a 6-cycle reset strategy.

We have presented results of empirical studies which demonstrate a competitive advantage of the penalty-based strategy over the Breakout strategy by comparing DBA with DisPeL. We have then justified DisPeL's high performance when compared to DBA.

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. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on ArtificialIntelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.
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. Fred Glover. Tabu search - part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.
5. 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.
6. 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.
7. Patrick Mills. *Extensions to Guided Local Search*. PhD thesis, University of Essex, Essex, 2002.
8. Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45, 1995.
9. John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of AAAI'04*, pages 191–196, 2004.
10. Dave Tompkins and Holger Hoos. Warped landscapes and random acts of SAT solving. In *8th International Symposium on Artificial Intelligence and Mathematics (AMAI 2004)*, January 2004.
11. 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.
12. 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.