On the Behavior and Application of Constraint Weighting

John Thornton[1] and Abdul Sattar[2]

[1] School of Information Technology, Griffith University Gold Coast,
Parklands Drive, Southport, Qld, 4215, Australia
j.thornton@gu.edu.au
[2] School of Computing and Information Technology, Griffith University,
Kessels Road, Nathan, Qld, 4111, Australia
sattar@cit.gu.edu.au

Abstract. In this paper we compare the performance of three constraint weighting schemes with one of the latest and fastest WSAT heuristics: rnovelty. We extend previous results from satisfiability testing by looking at the broader domain of constraint satisfaction and test for differences in performance using randomly generated problems and problems based on realistic situations and assumptions. We find constraint weighting produces fairly consistent behaviour within problem domains, and is more influenced by the number and interconnectedness of constraints than the realism or randomness of a problem. We conclude that constraint weighting is better suited to smaller structured problems, where it is can clearly distinguish between different constraint groups.

1 Introduction

The intensive research into satisfiability testing during the 1990s has produced a set of powerful new local search heuristics. Starting from GSAT [11], the latest WSAT techniques have raised the ceiling on solving hard 3-SAT problems from several hundred to several thousand variables [13]. At the same time, a new class of clause or constraint weighting algorithms have been developed [9, 12]. These algorithms have proved highly competitive with GSAT (at least on smaller problems with few solutions [1]), and have stimulated the successful application of related techniques in several other domains [14, 2, 15]. However, since the initial development of constraint weighting, WSAT has evolved new and more powerful heuristics to meet the challenges posed by planning problems (such as novelty and rnovelty [8]). The improved performance of these heuristics brings the usefulness of constraint weighting into question. This paper re-examines constraint weighting in the light of the latest WSAT developments. We take the basic heuristics of both techniques and apply them to a series of different problem domains. In the process we examine the following questions:

- Are there particular problem domains for which constraint weighting is preferred?
- Does constraint weighting do better on more realistic, structured problems?
- Is there one weighting scheme that is superior on all the domains considered?

The main aim of the study is to provide practical guidance as to the relevance and applicability of constraint weighting to the broader domain of constraint satisfaction. Research has already looked at applying WSAT to integer optimization problems [16], and applying constraint weighting to over-constrained problems (CSPs) [15]. However, outside the satisfiability domain, there has been little direct comparison between techniques. The research addresses this by applying both WSAT and constraint weighting to three CSP formulations: university timetabling, nurse scheduling and random binary constraint satisfaction. In addition we explore the behavior of constraint weighting on several classes of satisfiability problem, and look at the performance of a hybrid WSAT + constraint weighting algorithm.

The next section introduces the algorithms used in the study, and then the results for each problem domain are presented. From an analysis of these results we draw general conclusions about the applicability and typical behavior of constraint weighting.

2 CSPs and Local Search

The constraint satisfaction paradigm models the world in terms of variables with domains of values, and constraints that define allowable combinations of these values [7]. One of the aims of constraint satisfaction is to provide a uniform way of representing a range of problems that can then be solved using standard CSP techniques. Local search becomes relevant to constraint satisfaction when problems become too large or complex to solve using a systematic technique such as backtracking [6]. Instead of building up a solution by instantiating variables one at a time, local search starts with a complete instantiation of all variables and searches for improving 'local' moves. Generally this means trying domain values for each variable and accepting the value that most reduces the overall cost. Repeatedly making improving moves will either lead to a solution (all constraints satisfied) or to a local minimum (some constraints violated, but no more improving moves available). The basic issue for all non-trivial local search techniques is how to escape local minima and carry on the search.

2.1 Constraint Weighting

Constraint weighting schemes solve the problem of local minima by adding weights to the cost of violated constraints. These weights permanently increase

the cost of violating a constraint, changing the shape of the cost surface so that minima can be avoided or exceeded [9].

```
procedure ConstraintWeighting
begin
    CurrentState ← set variables to initial assignments
    BestCost ← cost of CurrentState
    while BestCost > 0 and iterations < MaxIterations do
        if MOVE or CurrentState is not a local minimum then
            Dlist ← Empty
            select a variable v_i involved in a constraint violation
            for each domain value d_j of v_i | d_j ≠ current value of v_i do
                TestCost ← cost of accepting d_j
                if TestCost ≤ BestCost then
                    if TestCost < BestCost then
                        Dlist ← Empty, BestCost ← TestCost
                    end if
                    Dlist ← Dlist + d_j
                end if
            end for
            if Dlist not Empty then
                CurrentState ← randomly accept move from Dlist
            else if MOVE then
                increment the weight of all violated constraints containing v_I
                BestCost ← new cost of CurrentState
            end if
        else if MIN then
            increment the weight of all violated constraints
            BestCost ← new cost of CurrentState
        else if UTIL then
            increment the weight of all violated constraints with the smallest weight
            BestCost ← new cost of CurrentState
        end if
    end while
end
```

Fig. 1. The Constraint Weighting Algorithm

Several weighting schemes have been proposed. In Morris's [9] formulation, constraint weights are initialised to one and violated constraint weights are incremented by one each time a local minimum is encountered. Frank [3, 4] adjusts weights after each move and experiments with different initial weights and weight increment functions and with allowing weights to decay over time. Further work has applied constraint weighting to over-constrained problems using dynamic weight adjustment [14] and utility functions [15].
In the current study we are interested in *when* and *what* to weight. Therefore we keep to Morris's original incrementing scheme and explore variations of three of the published weighting strategies:

1. MIN: Incrementing weights at each local minimum (based on [9]).

2. MOVE: Incrementing weights when no local cost improving move exists (based on [3], although Frank increments after all moves).
3. UTIL: Incrementing weights at each local minimum according to a utility function (based on [15]).

Voudouris and Tsang's [15] utility function is part of a more sophisticated algorithm (Guided Local Search or GLS) that handles constraints with different absolute violation costs. They penalise features in a local minimum that have the highest utility according to the following function:

$$utility_i(s^*) = I_i(s^*) \times (c_i / (1 + p_i))$$

where s^* is the current solution, i identifies a feature, c_i is the cost of feature i, p_i is the penalty (or weight) currently applied to feature i and $I_i(s^*)$ is a function that returns one if feature i is exhibited in solution s^* (zero otherwise). In the current study we assume each feature is a constraint with a cost of one. In this case the utility function will only select for weighting the violated constraint(s) in a local minimum that have the smallest current weight. Our aim is to test the utility function as a weighting strategy in isolation from the GLS algorithm, to see if it is useful in a more general weighting approach.

The three weighting strategies are tested within the same basic algorithm which randomly selects variables involved in constraint violations and accepts the best non-cost increasing move from the variable's domain (see Fig. 1). This algorithm was adjusted for SAT problems to randomly select violated *clauses* (rather than variables) and then accept the best non-cost increasing move from the combined domains of all variables involved in the clause (following the WSAT approach of Fig. 2). In addition, the MOVE heuristic was adapted for SAT to increment the weight of a selected clause if no improving move exists for that clause. These strategies were chosen due to the special structure of SAT problems in that all variables have two domain values (true or false), which would otherwise cause the original Fig. 1 algorithm to only consider a single move in each iteration.

2.2 WSAT

WSAT avoids local minima by allowing cost *increasing* moves. The algorithm proceeds by selecting violated constraints and then choosing a move which will improve or satisfy the constraint, even when this results in an overall cost increase. The various WSAT schemes differ according to the move selection heuristic employed. The rnovelty heuristic [8] considers both the overall cost of a move and when the move was *last* used. If the best cost move is also the most recently used move then (according to a probability threshold) the second best cost move may be accepted (as shown in figure 2). In using a least recently used comparison, rnovelty combines a stochastic search with a memory strategy similar to that proposed for HSAT [5]. It is the extra leverage obtained by considering the age of a move that distinguishes rnovelty from the earlier WSAT heuristics.

```
procedure rnovelty
begin
    CurrentState ← set variables to initial assignments
    while Cost(CurrentState) > 0 and iterations < MaxIters do
        if iterations modulus 100 = 0 then
            CurrentState ← CurrentState + random move
        end if
        select a violated constraint c
        Best ← n, SecondBest ← n | Cost(n) = LargeValue
        LastChange ← last iteration variable in c was changed
        for each variable v_i involved in c do
            for each domain value d_j of v_i | d_j ≠ current value of v_i do
                if Cost(d_j) < Cost(Best) or (Cost(d_j) = Cost(Best)
                    and LastUsed(d_j) < LastUsed(Best)) then
                        SecondBest ← Best, Best ← d_j
                else if Cost(d_j) < Cost(SecondBest) or
                    (Cost(d_j) = Cost(SecondBest) and
                    LastUsed(d_j ) < LastUsed(SecondBest)) then
                        SecondBest ← d_j
                end if
            end for
            if LastUsed(Best) = LastChanged and
                (Cost(Best) - Cost(SecondBest) < MinDiff or
                random value < NoiseParameter) then
                    Best ← SecondBest
            end if
            CurrentState ← CurrentState + Best
        end for
    end while
end
```

Fig. 2. The rnovelty heuristic

3 Experimental Results

3.1 Satisfiability Results

Research has already demonstrated the superiority of constraint weighting over GSAT and early versions of WalkSAT for smaller randomly generated 3-SAT problems (up to 400 variables) and for single solution AIM generated problems [1]. To see if these earlier results still hold, we updated Cha and Iwama's study by comparing our constraint weighting algorithms with McAllester *et al.*'s WSAT implementation of rnovelty [8]. For our problem set we randomly generated 100, 200 and 400 variable 3-SAT problems with a clause/variable ratio of 4.3 and selected ten satisfiable problems for each problem size (shown as r100, r200 and r400 in table 1). At each problem size we calculated the average of 100 runs. We also used the 4 AIM generated single solution 100 variable problems available from the DIMACS benchmark set (see ftp://dimacs.rutgers.-

edu/pub/challenge/sat/benchmarks/cnf). Each problem was solved 100 times and the average for all 4 problems reported. Table 1 shows the results for these problems[1] and confirms constraint weighting's superiority for small AIM formula, but indicates rnovelty has better random 3-SAT performance. The results also show there is a growing difference between constraint weighting and rnovelty as the problem size increases. For the r400 problems rnovelty is still solving 98% of instances within 1,000,000 flips where the success rate for the best weighting strategy (MOVE) has dropped to 84%. Of the weighting strategies, MOVE outperforms MIN on most problem sizes (except r100), while MIN does somewhat better than UTIL (although the gap closes for the larger problems). In table 1, Max-Flips is the number of flips at which unsuccessful runs were terminated, average flips is the average number of flips for *successful* runs, time is the average time for *successful* runs and success is the percentage of problems solved at Max-Flips.

Table 1. Results for small 3-SAT problems

Problem	Max Flips	Constraints	Method	Average Flips	Time (secs)	Success
r100	250000	432	rnovelty	1165	0.03	100%
			MIN	2221	0.04	100%
			MOVE	2162	0.06	100%
			UTIL	11285	0.22	98%
r200	500000	864	rnovelty	5319	0.11	100%
			MOVE	42043	1.27	91%
			MIN	45799	0.90	74%
			UTIL	42512	0.79	64%
r400	1000000	1728	rnovelty	69223	1.42	98%
			MOVE	180168	3.23	84%
			MIN	128243	2.67	34%
			UTIL	114408	2.20	31%
AIM 100	100000	200	MOVE	3800	0.06	100%
			MIN	6767	0.09	100%
			UTIL	42457	0.48	81%
			rnovelty	-	-	0%

To investigate the gap between constraint weighting and rnovelty for larger problems, we looked at the relative performance of the two algorithms for the DIMACS benchmark large 3-SAT problems (800 to 6400 variables). The graph in figure 3 shows the best result obtained for each algorithm (after 10 runs of 4 million flips on each problem) and confirms that constraint weighting performance starts to degrade as problem size increases. However, an interesting effect is that the relative performance of UTIL starts to improve as the problem size grows until it significantly dominates the other weighting methods.

To test whether the random 3-SAT results are reproduced in more structured domains we looked at a selection of conjunctive normal form (CNF) encodings

[1]All problems were solved on a Sun Creator 3D-2000

of large realistic problems (again from the DIMACS benchmark). We found rnovelty to dominate the large and hard graph coloring problems (g) where as constraint weighting performed better on the smaller circuit fault analysis (ssa), parity function learning and inductive inference problems (ii32). Representative results averaging 100 runs on selected problems are given in table 2:

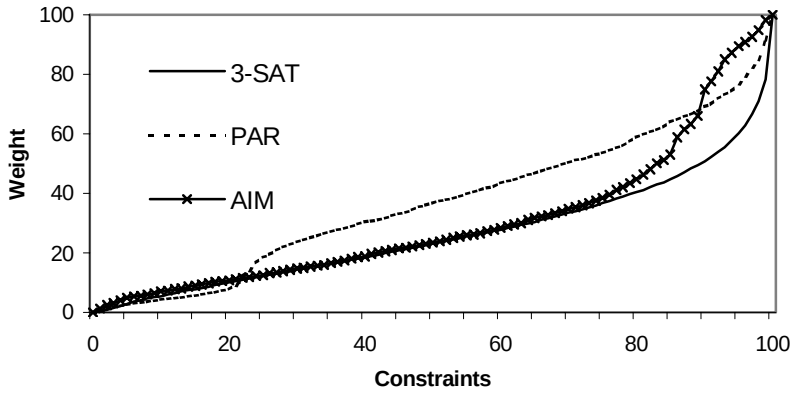

Fig. 3. Result plot for large DIMACS 3-SAT problems

Table 2. Results for large structured DIMACS problems

Problems	Max-Flips	Constraints	Method	Average Flips	Time(secs)	Success
g125.18	1000000	152064	rnovelty	5679	1.54	100%
g250.15			UTIL	160094	18.75	100%
Graph			MIN	218078	26.63	90%
Colouring			MOVE	222658	38.27	90%
ssa038-160	500000	3032	MOVE	2911	0.05	100%
Circuit			MIN	3182	0.08	100%
Fault			UTIL	17501	0.26	100%
Diagnosis			rnovelty	41897	0.97	100%
par8-2/4-c	250000	268	MOVE	1972	0.04	100%
Parity			rnovelty	2844	0.05	100%
Function			MIN	3981	0.06	100%
Learning			UTIL	12698	0.19	98%
ii32b3–e3	500000	8376	MIN	1125	0.25	100%
Inductive			UTIL	1241	0.27	100%
Inference			MOVE	2734	0.86	100%
			rnovelty	16097	3.30	100%

The overall DIMACS problem results show that constraint weighting does perform better on smaller, realistic (possibly structured) problems in comparison to rnovelty. The relative performance between weighting strategies is similar to the 3-SAT results, in that UTIL performs better on the larger problems (g), with MOVE dominating the majority of other problems (excepting ii32). However,

rnovelty's superiority on the graph problems again suggests weighting performs less well in longer term searches, as in all domains where constraint weighting dominates, (AIM, par, ii32 and ssa) solutions are found relatively quickly. In the large *and* difficult problems (e.g. 3-SAT and graph coloring) the constraint weighting heuristics do not seem to provide effective long term guidance.

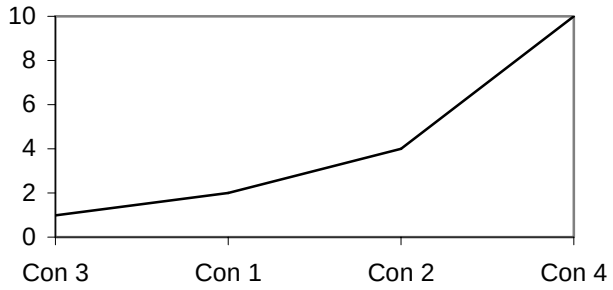

Fig. 4. An example constraint weight curve

Examining Constraint Weight Behavior. To further investigate the behavior of constraint weighting, we looked at the way constraint weights are built up during a search. To do this we developed *constraint weight curves* which plot the constraint weights on the y-axis and order the constraints on the x-axis according to their ascending weight values. For example, if at the solution point of a problem containing 4 constraints, constraint 1 has a weight of 2, constraint 2 has a weight of 4, constraint 3 has a weight of 1 and constraint 4 has a weight of 10, we would produce the graph in figure 4. In this way a picture is generated of the distribution of weights across the constraints.

Figure 5 shows the average weight curves for each 3-SAT problem size using the MOVE heuristic (normalized on axes from 0 to 100). We plotted additional curves for the larger 3-SAT problems and found that after an initial predictable adjustment period, curves very similar to those in figure 5 are produced. The other weighting strategies also produce consistent and similar curves, showing that in the longer term, the weighting process mainly serves to smooth the curves to an underlying distribution (for 3-SAT, curves of the form $y = a - b\log_n(c - x)$ provide a close fit).

To further explore this phenomenon we looked at curves for problems which constraint weighting found relatively easy to solve. Figure 6 shows the averaged MOVE curves for the DIMACS par and AIM problems and figure 7 shows the curves for the ii32, graph and ssa problems. In both figures an averaged 3-SAT curve is provided for comparison. The first feature observed from these graphs is that there is a high degree of consistency for the same problems but noticeable differences between domains. The graphs also show two distinct types of constraint weight curve: the figure 7 curves are similar (or uniform) in that after an initial steeper start all the curves have a steadily increasing gradient. These curves differ mainly in the steepness of their ascent and in the point at which the curve reaches the constraint axis. The curves in figure 6 show different behaviour, in that each curve deviates from the steadily increasing gradients of the other problems and exhibits some irregularity. For instance, the par problems

show a ‘step’ between 22 and 28 on the constraint axis and the AIM problems show a similar step between 80 and 95.

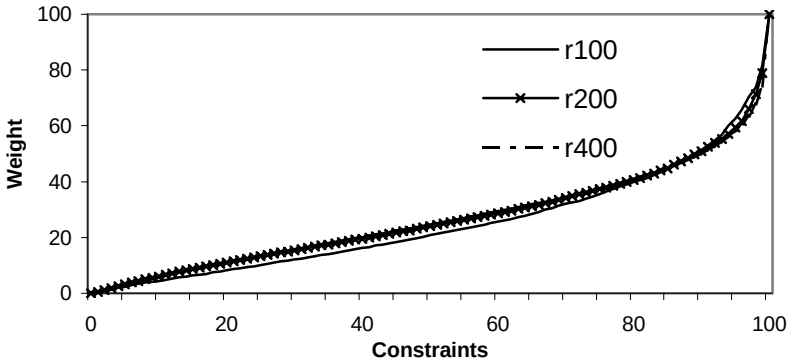

Fig. 5. Constraint weight curves for various 3-SAT problems

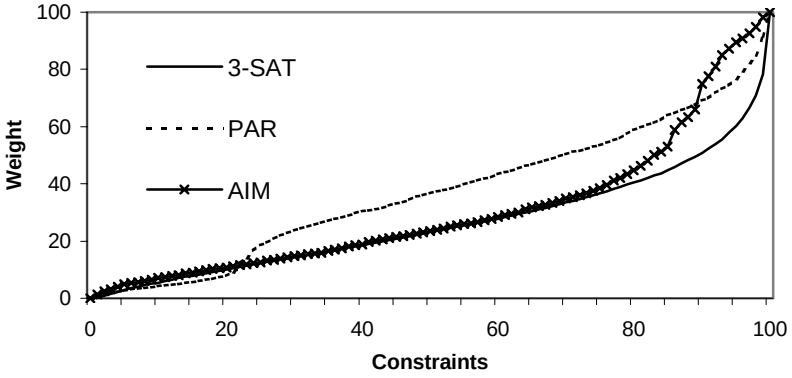

Fig.6. Constraint weight curves for the DIMACS par and AIM problems

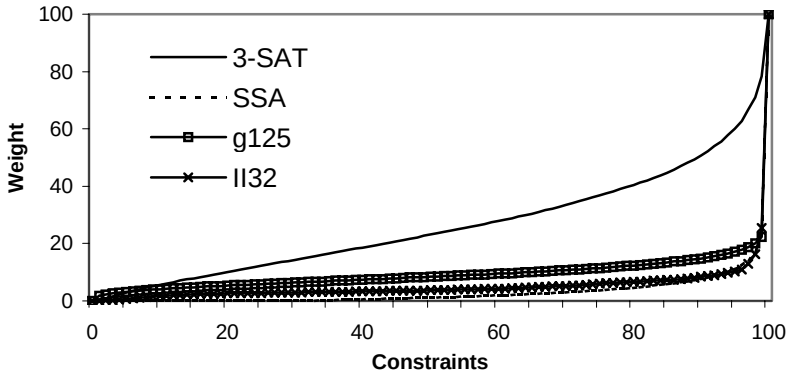

Fig.7. Constraint weight curves for the DIMACS graph, ssa and ii32 problems

Adding Weights to rnovelty. Given that no one technique proved better on all the satisfiability problems, we decided to explore a hybrid rnovelty algorithm with constraint weighting guidance added in. The rnovelty heuristic was changed by weighting each selected constraint iff the selected move for the constraint causes an overall (weighted) cost increase. Move evaluation is then based on the weighted cost. The new algorithm improved rnovelty's performance on all problems where constraint weighting previously dominated (AIM, par, ssa and ii32) but was unable to reach constraint weighting's original performance levels. For those problems where rnovelty dominated (3-SAT and graph coloring), the hybrid algorithm was still superior to constraint weighting, but not as good as pure rnovelty. In no instance was the average performance of the hybrid superior to both of the original techniques. For this reason, further exploration of the algorithm was rejected. An alternative technique which added an rnovelty value selection heuristic into constraint weighting was also implemented. This proved slightly inferior to the rnovelty hybrid and was also rejected.

3.2 CSP Results

Satisfiability is a subset of the broader domain of constraint satisfaction. Although CNF formulations can model multiple problem domains, they all share the same constraint type (i.e. clauses of disjunct literals). For many CSPs there are more natural and efficient ways of modeling constraints and variables. It is therefore significant to explore the performance of constraint weighting and rnovelty on a broader range of problems (also, to our knowledge, this is the first time rnovelty has been applied to CSPs). To this end, we look at two CSP formulations of real-world problems (university timetabling and nurse scheduling), both involving complex non-binary constraints and large non-standard variable domains. In addition we run tests on the well-studied problem of random binary constraint satisfaction [10].

For the purpose of the research, a university timetable problem generator was developed. The generator can be tuned to produce a wide range of realistic problems, while also having a mode that creates relatively unstructured, randomised problems. We were interested in building identical sized problem pairs, one reflecting the structure of a realistic timetabling problem (i.e. students doing degrees, following predictable lines of study, etc.) and the other using purely random allocations. The motivation was to test if a realistic problem structure influences the relative performance of the two algorithms.

The nurse scheduling experiments were run on a set of benchmark problems, taken from a real hospital situation. Each schedule involves up to 30 nurses, over a 14 day period, with non-trivial constraints defining the actual conditions operating in the hospital (for more details, see [14]).

Finally two sets of hard random binary CSPs were generated, each with 30 variables of domain size 10, and constraint density (p1) and tightness (p2) values that placed them in the accepted phase transition area [10]. This made the problems large and difficult enough to challenge the standard backtracking techniques.

Table 3 shows the results of running each class of problem against our four algorithms (all results are averages of 10 runs on 10 different problems). We

also report results for the binary CSPs using Van Beek's backtracking algorithm (see ftp://ftp.cs.ualberta.ca/pub/vanbeek/software). The table 3 results put rnovelty firmly ahead for the binary CSPs and slightly ahead for both classes of timetable problem, but strongly favor constraint weighting for the nurse scheduling problems. Adding structure to the timetabling problems does slow performance, but does not favor a particular method. As before, we are interested in the constraint weighting behavior, so figure 9 shows the average constraint weight curves for each class of CSP.

Table 3. Results for the CSP problems
(tt-struct = structured timetabling, tt-rand = random timetabling,
n-sched = nurse scheduling, binary = binary CSP)

Problem	Cut-off(secs)	Constraints	Method	Av. Iterations	Time (secs)	Success
tt-struct	500	500	rnovelty	306546	98.9	78%
			MIN	287874	101.3	74%
			MOVE	348327	164.9	70%
			UTIL	439288	177.4	62%
tt-rand	500	500	rnovelty	106357	28.7	95%
			MIN	111957	30.2	95%
			MOVE	113427	34.0	97%
			UTIL	187339	40.5	95%
n_sched	180	500	MIN	134303	57.7	94%
			MOVE	236142	103.3	81%
			UTIL	249317	108.6	81%
			rnovelty	1010629	99.4	77%
binary	350	200	rnovelty	103155	1.2	100%
n=30		200	MOVE	469329	5.2	93%
m=10		200	MIN	270339	3.3	79%
p1=80		200	UTIL	268949	3.4	75%
p2=17		1000	Backtrack	2.4×10^9	408.6	80%
binary	175	200	rnovelty	198833	1.4	100%
n=30		200	MOVE	239287	1.5	81%
m=10		200	MIN	254531	1.7	55%
p1=40		200	UTIL	226520	1.6	57%
p2=32		1000	Backtrack	33923486	16.4	100%

4 Analysis

The only direct way in which one constraint can influence another is by sharing variables. Hence we would expect problems with a high degree of interconnectedness between constraints to also form large difficult constraint groups. Random hard problem generators (as in 3-SAT and binary CSP) adjust the degree of connectedness to the critical level where the expected number of problem solutions approaches 1 [10]. Consequently, we expect such problems to exhibit an

upper bound on the *average* degree of connectedness for which problem solutions are possible. The 3-SAT and binary CSP constraint curves in figures 5 and 9 show that on these hard problems, constraint weights become spread across nearly all the problem constraints. The curves are also very similar in shape, exhibiting (after an initially steeper start) a constantly increasing gradient. Combining this information suggests that nearly *all* the constraints in these hard random problems belong to a single difficult constraint group. Although some constraints consistently accrue more weight than others, there is no *separation point* where a weighting algorithm can recognise that one constraint group is significantly different from another. This lack of distinction between constraint groups could explain constraint weighting's poorer performance on the 3-SAT and binary CSP random problems.

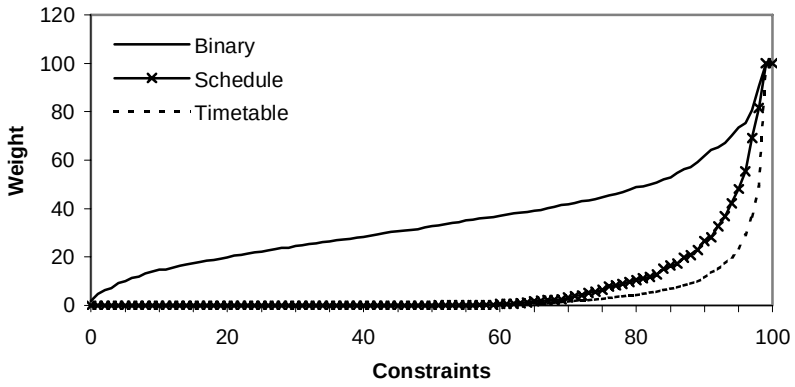

Fig.9. Constraint weight curves for the CSP problems

AIM and par curves. The constraint weight curves for the AIM and par problems in figure 6 diverge from the monotonically increasing curves of the other problems. These irregular curves indicate the weighting algorithm has found some difference between constraint groups. A closer inspection of the AIM problem structure supports this reasoning, as nearly all AIM variables appear in exactly 6 clauses. This tends to create tightly connected constraint groups that are relatively disconnected from the rest of the problem. Weighting can exploit this situation by ensuring the constraints in a harder group are kept true (by frequently placing weights on them) and then exploring the search space by violating constraints in the easier constraint groups. In doing this, constraint weighting will fix potential bottlenecks early in the search and quickly move to potential solution areas (in much the same way as a human problem solver). In contrast, non-weighting methods do not distinguish moves that violate difficult constraint groups and so are more likely to move into constraint violations that are harder to repair.

The parity learning function (par) curve in figure 6 also has an irregular shape, with a bulge appearing in the early part of the graph. This bulge indicates a group of relatively easy constraints that accrue less weight during the search, and offers an explanation of constraint weighting's relatively good performance on these problems. An examination of the par clauses also shows a structure of many small groups

of tightly connected constraints (both the AIM and par problems were artificially constructed to have a guaranteed solution). In combination, the 3-SAT, AIM and par results suggest that a non-uniform constraint graph (i.e. one that deviates from a 3-SAT type curve) can be an indication of an 'engineered' problem, i.e. one that exhibits a certain degree of regularity or structure. If this structure causes a significantly greater or lesser degree of interconnectedness amongst constraint sub-groups then we also expect that a constraint weighting algorithm will exploit this situation and gain a certain amount of leverage of other non-weighting techniques.

Graph colouring, ssa and ii32 curves. Figure 7 shows the constraint weight curves for the larger DIMACS problems: circuit fault diagnosis (ssa), inductive inference (ii32) and graph colouring. In comparison to 3-SAT, these curves show a similar monotonic increase but start with a shallower gradient and then have a distinct turning point where the gradient becomes noticeably steeper. If we consider the performance of constraint weighting on these problems, we find that a shallower initial gradient correlates with relatively better constraint weight performance. This can be explained by the lower curves indicating a greater weight distinction between the easy and hard constraints in the problem. Following our earlier reasoning, constraint weighting can exploit this difference by tending to avoid moves that violate heavily weighted constraints, and exploring violations of relatively easier constraints. The poorer performance of constraint weighting on the graph colouring problems can be further explained by the larger size of these problems and consequent scaling effects discussed later.

CSP curves. In looking at constraint weight curves for the more realistic CSP problems (nurse scheduling and timetabling), familiar smoothly increasing gradients are observed (see figure 9). However, for both problem types, approximately 60% of constraints accrue virtually zero weight. This contrasts with the similar curves in figure 7 where at least some weight has accrued on nearly all the constraints. Zero weight constraints are unlikely to provide leverage to a constraint weighting algorithm, because such constraints are *hard to violate*. Consequently, weighting and non-weighting algorithms will both tend to search in the space where these constraints are satisfied. Following this line of reasoning, we would not expect constraint weighting to have an advantage on the timetabling or nurse scheduling problems, whereas, in practice, MIN significantly outperforms the other techniques on the nurse scheduling problems. To investigate this further we looked at the *individual* constraint weight curves for each timetabling and nurse scheduling problem (figure 9 shows the *average* curves for 10 different problems). An initial inspection showed the nurse scheduling curves exhibited more irregularity of shape than the timetabling curves. To quantify this, we took the standard deviation of the weight value for each curve at each of the 100 points on the constraint axis and calculated the sum of these values for each problem type. This produced a summed deviation of 190.8 for the timetabling curves and 358.7 for the nurse scheduling curves (confirming the visual inspection). In addition, the shapes of the nurse scheduling curves

indicated that, in many cases, the constraint weighting algorithm was able to distinguish between constraint groups. Combining this information suggests that, although the constraint weight curves for timetabling and nurse scheduling were similar, the underlying behaviour of constraint weighting on the two problems was different and that constraint weighting was able to exploit irregularities in the nurse scheduling problems.

Scaling Effects. Given that constraint weighting does better when it finds distinctions between groups of constraints, we would expect the probability of randomly generating highly interconnected constraint groups to decline (causing the performance of constraint weighting to also decline) as problem size increases. For example, the number of constraints in a random 3-SAT problem grows at the rate of $4.32n$ (where n is the number of variables) whereas the number of *possible* constraints grows at a faster rate of $2n(n\text{-}1)(n\text{-}2)$. In addition there may be a *granularity* effect in constraint weighting, i.e. as the number of problem constraints increase, the effect of weighting a single constraint necessarily decreases, and constraints start to become weighted and violated in larger and larger groups. In this way the weight guidance becomes more general and less detailed, which could then cause promising search areas to be ignored. This is further backed up by the relative improvement in the performance of UTIL for longer searches: as UTIL increments weights less frequently than the other methods we would expect it's performance to deteriorate more slowly. This result ties in with Frank's work [4] on causing weights to decay during the search, and it may prove useful to investigate a combination of these strategies for larger problems.

5 Conclusions

The main conclusion of the paper is that constraint weighting is best suited to problems where the weighting process is able to distinguish sub-groups of constraints that have a distinctly different weight profile from the remaining problem constraints. We term such problems as having a structure which can be recognised either through an analysis of constraint weight curves or through a direct analysis of the interconnections of variables and constraints. Constraint weighting performance is therefore most influenced by the number and degree of interconnectedness of constraints rather than the degree of realism used in the problem generation. In addition, we found constraint weighting performance starts to degrade as problem size grows, due to a combination of larger problems tending to have less structure and a postulated weighting granularity effect. Additional experiments with hybrid constraint weight and rnovelty algorithms did not yield promising results, so further work in this area was rejected. Of the different weight heuristics, MOVE performed well on the binary CSP and satisfiability problems, while MIN did better on structured CSPs. UTIL was the best weighting strategy for larger problems, but was unable to match the performance of rnovelty. The paper also introduces constraint weight curves as a

method for analysing and predicting the behaviour of constraint weight algorithms. In our ongoing work, an extended version of this paper will present a larger sample of results and explore and compare the behavior of several competing constraint weight, WSAT and tabu search heuristics.

References

1. B. Cha and K. Iwama. Performance test of local search algorithms using new types of random CNF formulas. *Proceedings of IJCAI-95*, pages 304-310, 1995.
2. A. Davenport, E. Tsang, C. Wang and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. *Proceedings of AAAI-94*, pages 325-330, 1994.
3. J. Frank. Weighting for Godot: Learning heuristics for GSAT. *Proceedings of AAAI-96*, pages 338-343, 1996.
4. J. Frank. Learning short-term weights for GSAT. *Proceedings of IJCAI-97*, pages 384-389, 1997.
5. I. Gent and T. Walsh. Towards an Understanding of Hill-climbing Procedures for SAT. *Proceedings of AAAI-93*, pages 28-33, 1993.
6. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 32-43, Spring 1992.
7. A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99-118, 1977.
8. D. McAllester, B. Selman and H. Kautz. Evidence for invariants in local search. *Proceedings of AAAI-97*, pages 321-326, 1997.
9. P. Morris. The Breakout method for escaping from local minima. *Proceedings of AAAI-93*, pages 40-45, 1993.
10. P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):81-111, March 1996.
11. B. Selman, H. Levesque and D. McAllester. A new method for solving hard satisfiability problems. *Proceedings of AAAI-92*, pages 440-446, 1992.
12. B. Selman and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. *Proceedings of IJCAI-93*, pages 290-295, 1993.
13. B. Selman, H. Kautz and D. McAllester. Ten challenges in propositional reasoning and search. *Proceedings of IJCAI-97*, pages 50-54, 1997.
14. J. Thornton and A. Sattar. Dynamic Constraint Weighting for Over-Constrained Problems. *Proceedings of PRICAI-98*, pages 377-388, 1998.
15. C. Voudouris and E. Tsang. Partial Constraint Satisfaction Problems and Guided Local Search. *Proceedings of Practical Application of Constraint Technology (PACT'96)*, pages 337-356, 1996.
16. J. Walser, R. Iyer and N. Venkatasubramanyan. An integer local search method with application to capacitated production planning. *Proceedings of AAAI-98*, pages 373-379, 1998.