Learning from Failure in Constraint Satisfaction Search *

Diarmuid Grimes and Richard J. Wallace
Cork Constraint Computation Centre and Department of Computer Science
University College Cork, Cork, Ireland
{d.grimes, r.wallace}@4c.ucc.ie

Abstract

Much work has been done on learning from failure in search to boost solving of combinatorial problems, such as clause-learning in boolean satisfiability (SAT), nogood and explanation-based learning, and constraint weighting in constraint satisfaction problems (CSPs), etc. Many of the top solvers in SAT use clause learning to good effect. A similar approach (nogood learning) has not had as large an impact in CSPs. Constraint weighting is a less fine grained approach where the information learnt gives an approximation as to which variables may be the sources of greatest contention. In this paper we present a method for learning from search using restarts, in order to identify these critical variables in a given constraint satisfaction problem, prior to solving. Our method is based on the conflict-directed heuristic (*weighted-degree heuristic*) introduced by Boussemart et al. and is aimed at producing a better-informed version of the heuristic by gathering information through restarting and *probing* of the search space prior to solving, while minimising the overhead of these restarts/probes. We show that random probing of the search space can boost the heuristics power by improving early decisions in search. We also provide an in-depth analysis of the effects of constraint weighting.

1. Introduction

At first blush, one might imagine that learning from failure is a hopeless endeavour. After all, animals and human beings appear to be constructed to learn not from failure but from success, a principle enshrined in the so-called Law of Effect. This is not surprising when one considers that for most tasks there are so many wrong ways to do them that it would be impossible to store all this information, and it would still probably not be enough to learn the correct behaviour.

Why then should we expect that we can learn from failure in the context of constraint satisfaction search? The reason is based on the Fail-First Principle of (Haralick & Elliott 1980). This principle states that, "In order to succeed, try first where you are most likely to fail." That is, in solving CSPs it is generally best to deal with variables that cause difficulty as early as possible. This is because, since every variable must be assigned a value in a full solution, there is no point in making assignments to easy parts of the problem and then undoing them repeatedly when it proves impossible to find consistent value assignments for the remaining, difficult variables. In this situation, we may indeed be able to "learn from failure", because there is something positive to learn: that certain variables are sources of peculiar difficulty in a problem, usually because they represent sources of contention, and for this reason search should start by assigning these variables before other variables are considered.

This perspective should be distinguished from no-good learning, which is a more straightforward instance of learning from failure. Although the latter is sometimes effective in restricting search, by recognising that a given search path will not lead to a solution, for CSP search it has not been as successful as its formal equivalent in SAT (clause learning) (Katsirelos & Bacchus 2005), perhaps for the same reasons that were noted above in connection with animal and human learning, *viz* that the information learned is too specific to provide effective guidance at later stages of search.

However, learning from failure in the present sense poses special problems. In particular, since interactions between constraints in a problem can be complex, sources of difficulty can sometimes only be determined by actually carrying out search (Joslin & Clements 1998). A means of dealing with this problem is embodied in the "weighted degree" heuristic proposed by (Boussemart *et al.* 2004). This heuristic depends on a strategy of associating weights with each constraint. Weights are initially set to 1 and are incremented by 1 every time that constraint removes the final value in a domain (causing a domain wipeout). Variables are then selected on the basis of the total weight associated with their constraints. This is, therefore, an adaptive heuristic that works to enhance any heuristic that uses a variable's degree. In this case, as search continues it begins to improve over what it would have done if the original heuristic were used alone. This strategy has been shown to perform well on a variety of difficult problems (Boussemart *et al.* 2004) (Lecoutre, Boussemart, & Hemery 2004) (Hulubei & O'Sullivan 2006).

However, using weighted degree in this way has potential limitations. The most obvious is that the heuristic uses the least amount of data when making its most important

*This work was supported by Science Foundation Ireland under Grant 00/PI.1/C075.

choices, i.e. the first few variable selections. Improvements in search are likely to be restricted to lower levels of the search tree; in particular, search may never return to the first few variables.

It may, therefore, be possible to enhance the effects of weighted degree by starting over once information in the form of weights has been obtained. Restarting search is a widely used technique for problem solving that has proven to be effective in a number of domains. In SAT, the solver Chaff (Moskewicz, Madigan, & Malik 2001) uses restarts with clause learning to provide a chance to change early decisions in view of the current problem state. This solver was shown to be at least an order of magnitude faster than existing public domain SAT solvers on large, difficult problems.

Here we suggest a similar approach for CSPs that combines a benefit of restarting (e.g. traversing different areas of the search space) with learning from failure, which may serve to direct search more intelligently after restarting. As with the weighted degree heuristic, the goal of this strategy is to choose which variable to instantiate at a given point in search. In this case, we hope to take better advantage of the Fail-First Principle in order to constrain the underlying search space more effectively.

The next section provides background and definitions. The following section describes the basic strategies used in this paper. The next section gives the experimental methods and the basic results of our experiments. The next section contains an analysis of weight changes under different strategies, to provide a more in-depth understanding of these methods. The section after that discusses issues involved in this type of learning. The next section describes related work. The last section outlines our conclusions and plans for future work in this area.

Background

A constraint satisfaction problem (CSP) can be represented as a tuple of the form (V, D, C) where: $V = \{V_1, ..., V_n\}$ is a set of variables which must be assigned values; $D = \{D_1, ..., D_n\}$ is the set of domains for those variables consisting of possible values which may be assigned to the variables; $C = \{C_1, ..., C_m\}$ is the set of constraints. These constraints express relations $Rel(C_j)$ among domain values that that can be assigned to the variables in the scope of the constraint, $(Vars(C_j))$. In this paper we are concerned with binary constraint problems, so a constraint has at most two variables in its scope. Two variables are said to be neighbors if there is a constraint between them, i.e. if $\exists\ C_j$ s.t. $(X_a, X_b) \in (Vars(C_j))$, then X_a and X_b are neighbors.

An *assignment* is a set of tuples $A = \{(V_1, a), (V_2, b), ..., (V_k, h)\}$, each tuple consisting of a different instantiated variable and the value that is assigned to it. A *solution* to a problem is an assignment $A = \{(V_1, a), (V_2, b), ..., (V_n, x)\}$ that includes all variables and that does not violate any constraint, i.e. an assignment of a value to every variable such that no constraint is violated.

A binary constraint satisfaction problem has an associated constraint graph, where a variable is represented by a node in the graph and a constraint C_j is represented by an edge between the two nodes in $Vars(C_j)$. When the variables at both endpoints of an edge are assigned a value consistent with the edge-constraint, that edge is removed from the dynamic constraint graph. CSPs are also characterized by certain basic parameters. Thus, the *domain size* of a variable is the number of values in its current domain, and the *degree* of a variable is the number of edges connected to it. A constraint satisfaction problem is said to be *arc-consistent* when every value in every domain has at least one support in the domain of every neighboring variable.

During simple backtrack search a value is assigned to each variable until either a complete consistent assignment has been found or a dead-end (in the form of an inconsistent assignment) has occurred. In the second case, search removes the last value assigned and tries a different value in the domain. If there are no other values left in the domain, search backtracks up a level to the previous instantiation and removes the value assigned from that variable's domain and tries another value.

Many methods have been suggested which improve on this simple method of search. One type of method is the use of propagation or look-ahead techniques. One can propagate using MAC (maintaining arc-consistency), where the problem is made arc-consistent after every assignment, i.e. all values which are arc-inconsistent with that assignment are removed from the current domain of their variable. In this way, effects of an assignment are propagated through the problem. Clearly only uninstantiated variables need to be checked for consistency with an assignment when using MAC since all values already assigned to variables must be consistent with the new value assigned.

Even though all variables will be part of the solution, the order in which variables are selected for instantiation during search can have a large effect on the effort required. Typically a variable is selected according to some heuristic metric obeying the Fail-First principle. One of the most widely used heuristics in CSP solving is that which minimises the value of $domain\ size/degree$ (domain/degree for short). By choosing the variable with the smallest domain, one is ensuring that if there is to be a failure one will have the fewest alternative values to try. Choosing the variable with the largest degree ensures that through propagation one will generally have the largest reduction in search space.

Description of the Algorithms

The basic weighted-degree heuristic was described in the Introduction. It should be noted, however, that when selecting a variable, the edge-weight-sum for each variable is based only on edges connecting it to uninstantiated variables. (The heuristic is, therefore, a weighted-forward-degree heuristic.) These sums are compared, and the variable with the largest sum is selected as the next variable for instantiation. In this work, we use a variant of the heuristic in which the weight information is incorporated into the minimum domain/forward-degree heuristic to produce a minimum domain/weighted-degree heuristic (dom/wtdg for short).

In the present work, we combined dom/wtdg with restart in two ways that represent fundamentally different strate-

gies for learning. In both methods, edge-weights are carried along from restart to restart. In the first method these weights are used by the dom/wtdg heuristic for variable selection throughout search. In the second method, variables are chosen at random during all but the last restart. During this time weights are incremented in the usual fashion, but they are not used to guide search. On the last restart (final run), the dom/wtdg heuristic is used, based on the accumulated weights for each variable, which continue to be updated during this run. This method, therefore, consists of an information-gathering or "probing" stage (runs 1 through $R-1$), followed by a search stage.

In both methods, restarts occur after a fixed number of nodes have been explored in the search tree ("fixed cutoff", C). In addition, there is a fixed maximum number of runs, R, which gives a maximum number of restarts = $R-1$. If a solution is found before the Rth run, the process terminates at that point. Otherwise, it continues until the final run, at which point the cutoff is removed (increased to ∞). This makes the method complete.

Thus, during each run, search continues until the problem has been solved or C nodes have been searched, while updating constraint weights after domain wipeouts. On runs 1 through $R-1$, if no solution has been found after searching C nodes, search restarts on the initial state of the problem with updated weights. On the Rth run, search runs to completion, i.e. it runs until it either solves the problem or proves the problem insoluble. On this run, the dom/wtdg heuristic is always used for variable selection.

The first method, restart with dom/wtdg, was tested with the expectations that:

- for the *easier* problems in the set, a solution may be found without restarting or after a small number of restarts (unless the cutoff is too low).
- for *harder* problems, this approach, which learns from each search attempt, may begin to solve problems within the cutoff once it has enough information to make good variable selections early in search.

The second method, search after probing for failure using random variable selection, was tested with the expectation that the probing procedure would give a better overall indication of points of difficulty within a problem, although it is unlikely to solve any problem instances before the Rth run.

Tests of Restart Strategies

Experimental Methods

The basic algorithm used in the experiments reported in this paper was a MAC-3 algorithm with d-way branching. In addition to the variable ordering heuristic described above, for the soluble problems a promise value ordering heuristic was used (based on (Geelen 1992)). In the version used in this work, the heuristic calculates, for each value in the domain of the current variable, the product of the supporting values in the domains of *neighboring* uninstantiated variables, and chooses as the next value the one with the largest product.

The experiments reported in this paper were carried out with random binary CSPs. The problem parameters are based on two of the problem sets used by (Boussemart *et al.* 2004) in their study of the weighted degree heuristic. The parameter values are the same as those used in that paper; the only difference is that each of the present problems consisted of a single connected component. This was done by first connecting the nodes of the constraint graph by a spanning tree and then adding edges at random until the required number had been chosen. The problem sets tested on are in the critical complexity region.

For the first problem set, we tested soluble and insoluble problems separately to see if our approach was better suited to one or the other class. This was done by generating several hundred problems, and then selecting the first 100 soluble and the first 100 insoluble problems from the original set. Due to time constraints, most of our experiments have been done with the 100 soluble problems, and these were used for experiments reported in the next two sub-sections. The parameters for these problem sets were <200, 10, 0.0153, 0.55>, i.e. problems had 200 variables, initial domain sizes were all 10, the constraint graph density was 0.0153 (500 constraints), and the tightness of each constraint was 0.55. The parameters for a second problem set, in which both soluble and insoluble instances appear, were <80, 10, 0.1042, 0.35>.

For experiments on dom/wtdg with restart, the cutoff levels used were 250, 500, 1000, and 2000 nodes. The number of restarts were 1, 5, 10 and 20. These *factors* were "fully crossed" in the experiments, so that all levels of the cutoff factor were tested with all levels of the restart factor. Thus, all combinations of factor-levels were tested. (For the insoluble problem set only one combination was tried, 10 restarts with a 500 node cutoff).

For the random probing for failure experiments, each condition, i.e. combination of levels for cutoff and restarts, was tested ten times. This was done to obtain adequate samples under randomisation and to avoid spuriously good (or bad) effects due to random selections.

Results were analysed with the analysis of variance (ANOVA), in some cases followed by comparisons of individual means using paired-comparison t-tests (Hays 1973). For experiments with dom/wtdg with restart (next subsection), the ANOVA was based on a two-way fixed effects model. Experiments on search after probing for failure (subsequent sub-section), were analysed with a three-way mixed effects model in which problems was a third factor (and each cell of the design had ten data points based on the ten tests for that problem), as well as a two-way ANOVA based on the mean for each problem across the ten tests.

To meet the statistical assumptions of the ANOVA, and to make computations tractable, the original data were transformed by taking logarithms prior to the analysis. (Since the main test problems are in the critical complexity region, heavy-tail effects are not present; in addition, for these problems such effects should be eliminated by performing arc consistency after each instantiation.) For the ANOVAs, the actual analyses were carried out 'by hand' in EXCEL, i.e. functions such as SUM and SUMSQ were used, but not the statistical tools provided by the program. The t-tests employed the paired-comparison test available in EXCEL.

Results for dom/wtdg with Restart

For reference, we first note the performance of the basic dom/wtdg heuristic on these problems. For the soluble problem set, this heuristic required a mean of 37,934 nodes and 94,162,635 constraint checks to find one solution. It is also of interest that for the soluble problem set, the basic min domain/forward-degree heuristic required a mean of 107,059 nodes. and 226,235,887 constraint checks; this confirms that dom/wtdg is a very powerful heuristic when used on these problems. (Since nodes, constraint checks, and runtime were all highly correlated in these experiments, for brevity, results will be reported in terms of nodes.)

Summary results for the tests of dom/wtdg with restart are given in Table 1. The table shows means for "total nodes" over the 100 (soluble) problems for each condition. This measure includes nodes searched across all ($\leq R$) runs, including the final run to completion if one was needed. For these data, there were no consistent trends for either the number of restarts or the cutoff value. In addition, the analysis of variance (ANOVA) showed no statistically significant differences for either the cutoff or restarts factors. This means that differences among the means in Table 1 are not evidence of reliable differences between conditions.

Table 1. Results with Repeated dom/wtdg
Total Search Nodes

restarts \ cutoff	250	500	1000	2000
1	49,185	52,974	40,375	48,189
5	45,283	52,222	45,773	47,574
10	46,648	46,525	42,165	53,483
20	53,262	47,656	51,811	52,160

Notes. <200,10,.0153,.55> problems.
Means for 100 problems.

In addition to the basic assessment of total effort, it was of interest to consider means for the last run for each problem, to determine if there was any evidence of improvement that could be ascribed to learning. These results are given in Table 2. Again, there are no consistent trends, although means tended to decrease as either the cutoff or number of restarts increases. In this case, however, the ANOVA gave statistically significant results for each factor, although not for their interaction. For restarts, $F(3,1584) = 7.627$, $p < 0.01$; for cutoff, $F(3,1584) = 10.953$, $p < 0.01$.

Table 2. Repeated dom/wtdg: Final Run

restarts \ cutoff	250	500	1000	2000
1	48,958	52,489	39,445	46,469
5	44,053	49,867	41,433	40,194
10	44,248	41,975	34,125	40,383
20	48,642	38,976	37,461	29,140

Notes. <200,10,.0153,.55> problems.
Mean nodes for 100 problems.

One possible reason for the statistically significant results that may confound interpretation is that a problem was only tested until a solution was found. This means that whenever a problem was solved before the $R-1$th restart, it would necessarily have a low value for the number of nodes in the final run. This would be more likely to happen for larger C and R, and is suggested by the overall trends noted above. This may have happened either because the weights were effective at this point or because a degree of randomisation was added to a heuristic that is generally effective. Therefore, cases where the mean was better than the reference value (37,934), which sometimes occurred when the number of restarts was 10 or 20 (see Table 2) must be treated with caution.

To obviate this difficulty, 39 problems were selected that were *never solved before the final run under any condition*. For these problems, dom/wtdg alone required a mean of 77,111 nodes to find a solution. The means for these 39 problems under the present experimental regimen are given in the table below. For this problem set, the ANOVA gave *no* statistically significant results. This indicates that improvements over ordinary dom/wtdg in Table 2 and the statistically significant results may be due to the confounding factor suggested above.

Table 3. Repeated dom/wtdg: Only Solved on Final Run

restarts \ cutoff	250	500	1000	2000
1	78,227	97,048	63,318	85,327
5	70,643	88,570	78,570	86,240
10	75,439	82,901	69,995	90,979
20	95,526	72,389	82,116	70,345

Notes. <200,10,.0153,.55> problems.
Mean nodes for 39 problems only solved on run to completion.

There are at least two reasons why dom/wtdg with restart may not be effective. Since failure always occurs in a particular context in the form of a partial assignment, weights derived from such failures may not be relevant at the beginning of search when the contexts of those failures are no longer present. Furthermore, variables with higher weights will tend to be chosen earlier after restart and, since wipeouts generally don't occur until after several variables have been assigned and only affect future variables when constraint propagation is used, these variables are less likely to have their weights increased. This will lead to different variables being weighted each time. When these variables are chosen during subsequent restarts, still other variables may be weighted, and so forth. Together, these effects will make it difficult to distinguish sources of "global" as opposed to "local" difficulty, i.e. difficulty that is basic to the problem and which will, therefore, be encountered throughout the search space, versus difficulty restricted to a specific part of the search space.

Results for Search after Probing for Failure

If variables are chosen at random during search, then information concerning problem difficulty can be obtained from a wider sample of the search space and is thus more likely to to be related to global difficulties. Although a given domain wipeout is still dependent on a specific assignment at that point, with repeated restarts it is likely that different parts of the search space will be explored. After a sequence of such restarts, a high constraint weight indicates that this constraint was directly linked with domain wipe-outs in many different parts of the search space, and is therefore associated with a basic source of difficulty in the problem.

Table 4. Results with Repeated Random Variable Selection - Total Search Nodes

		cutoff		
		250	500	1000
re-starts	1	44,130	48,123	47,204
	5	40,048	42,318	42,073
	10	37,179	36,192	40,503
	20		39,135	

Notes. <200,10,.0153,.55> problems.
Means over 10 runs with 100 problems.
Time did not permit collecting data for all conditions with 20 restarts.

Domain wipeouts that occur early in search are also more likely to reflect basic difficulties, since the smaller the assignment that produces a wipeout, the greater the probability that the constraint will cause a wipeout in a different part of the search space. (This is similar to non-generalised nogood learning, where more effective nogoods are those that are shorter in length). This implies that, for $C > n$, a shorter cutoff (C_i) with more restarts (R_i) should learn better information than a larger cutoff (C_j) with fewer restarts (R_j), where C_i x R_i = C_j x R_j, i.e. where the total number of search nodes while probing is the same.

These expectations are largely bourne out by the experiments using restart with random variable selection. In these tests, in contrast to those with restart using weighted degree, there was sometimes evidence of improvement even for total nodes (Table 4). More significantly, in the final runs search was clearly superior to the basic dom/wtdg heuristic (Table 5: 10 and 20 restarts). As already noted, due to the inefficiency of search with random variable ordering, $R - 1$ restarts were carried out for each problem under all conditions. Hence, the results in Table 4 are not affected by finding solutions before the cutoff prior to the last restart.

The ANOVA for this experiment (based on 1, 5, and 10 restarts) gave a highly significant result for the restarts factor; in contrast, the result for the cutoff factor was not significant (Table 6). Not surprisingly, the problems factor was highly significant, showing that there were stable differences in problem difficulty despite the randomisation of variable selection. The only significant interaction was between problems and number of restarts.

Table 5. Repeated Random Variable Selection: Final Run (Search Nodes)

		cutoff		
		250	500	1000
re-starts	1	43,880	47,623	46,204
	5	38,798	39,818	37,073
	10	34,679	31,192	30,503
	20		29,135	

Notes. <200,10,.0153,.55> problems.
Means over 10 runs with 100 problems.

For comparisons of means in Table 5, a second ANOVA was done with only the Restarts and Cutoff factors, using the means for the ten tests for each problem and condition. Again, the restarts factor was highly significant statistically ($F(2, 891) = 8.687$, $p < 0.01$), while the cutoff factor was not ($F(2, 891) = 0.313$). Post-hoc comparisons were made for the following pairs of conditions

1. 5-restarts/500-cutoff versus 10-restarts/250-cutoff
2. 5-restarts/1000-cutoff versus 10-restarts/500-cutoff

In both cases, the difference between means was statistically significant ($t(99) = 3.391$, $p < 0.001$, and $t(99) = 2.913$, $p < 0.01$, respectively, for the comparisons just noted [one-tail tests].) This supports the expectation that superior learning should result from shorter cutoffs and more restarts as compared to longer cutoffs with fewer restarts.

Table 6. Analysis of Variance for Repeated Variable Selection, Final Run

factor	SS	df	MS	F	p
Problems	2472.57	99	24.98	114.548	$<< 0.01$
Restart	38.69	2	19.35	88.726	$<< 0.01$
Cutoff	0.69	2	0.35	1.593	ns
P*R	97.79	198	0.49	2.265	< 0.01
P*C	45.29	198	0.23	1.049	ns
R*C	1.60	4	0.40	1.834	ns
P*R*C	83.52	396	0.21	0.967	ns
Error	1766.07	8100	0.22		

Notes. Based on data of Table 5, omitting data for 20 restarts. Column headers are standard abbreviations for ANOVA terms: "SS" = sum of squares, "df" = degrees of freedom, "MS" = mean sum of squares (SS/df), "F" is the test statistic obtained by dividing the MS in that row by the MS-Error. "p"is the probability of obtaining the value of F under the null hypothesis of no effects, where the expectation of F=1.

Results for Insoluble Problems

The basic dom/wtdg averaged 41,200 nodes in solving 100 problems. The repeated dom/wtdg with 10 restarts and a cutoff of 500 nodes did worse, averaging 47,072 nodes in total. For the random variable selection while probing, we again did ten experimental tests with 10 restarts and a cutoff of 500 nodes. This did better than basic dom/wtdg in total nodes on every run, averaging 39,312 nodes (so the final run averaged 34,312 nodes).

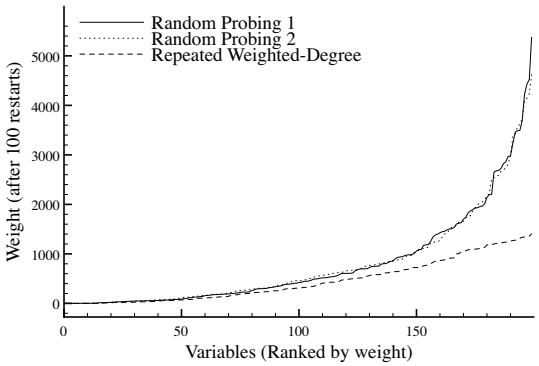

Figure 1: Variable weights after 100th restart, with C = 1000, on a sample problem instance. Two independent tests with random variable selection are shown together with results for repeated weighted degree.

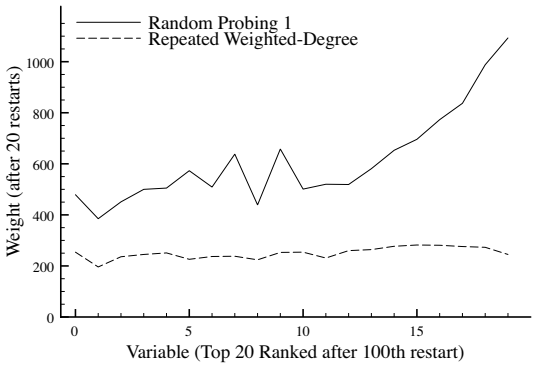

Figure 2: Variable weights after 20th restart on a sample problem instance. The twenty variables are ranked by their weight after the 100th restart (C = 1000).

Results for Mixed Problem Set

A lexical value ordering was used for these 100 problems, 46 of which were insoluble. The basic dom/wtdg averaged 186,854 nodes over the 100 problems. Repeated dom/wtdg with 20 restarts and a cutoff of 200 nodes did worse, averaging 220,727 nodes in total. For the random variable selection while probing, we again did ten experimental tests with 10 restarts and a cutoff of 500 nodes. This did better on average than basic dom/wtdg in total nodes, averaging 183,405 nodes (so the final run averaged 178,405 nodes).

Analysis of Weight Changes Produced by Each Strategy

To better understand the differences between the two strategies used in the previous experiments, we ran both random variable ordering and dom/wtdg variable ordering with 100 restarts on selected problems. For these tests, we used cutoffs of 250 and 1000 nodes. The five problems chosen for these tests were sufficiently difficult that they could not be solved within the specified cutoff. Each variable's cumulated edge-weight was saved after every restart.

After 100 restarts the sum of weights across all variables was 50% to 100% greater for random variable ordering than for repeated dom/wtdg, depending on the cutoff. This means that when using random ordering there were 50% to 100% more domain wipeouts than when using dom/wtdg.

One reasonable explanation for the greater effectiveness of random probing for failure is that it provides better discrimination among variables, especially those with the largest weights. Figure 1 shows a typical weight plot for a single problem after 100 restarts, for the repeated wtdeg experiment and for two random variable ordering experiments. In all three cases the variables were ranked according to their weight after the 100th restart. The slope of the line indicates the level of discrimination between successive variables. Note that for both random orderings the slope is very steep for the top 50 variables and maintains a better level of discrimination over the next 50 variables than repeated weighted-degree. The slope for repeated-weighted degree has a much more gradual inclination, which indicates that even after 100 restarts there are no variables with a clearly larger weight than their predecessor.

We evaluated the stability of a variable's ranking in terms of its edge-weights statistically, using Kendall's coefficient of concordance, W (Hays 1973). This statistic allows one to compare m distinct rankings in order to assess their overall similarity. (Possible values of the statistic range from 0 to 1, inclusive.) Five problems were tested, for each we selected the 20 variables that had the highest weights *after the final run*. (Since domains of low-weight variables are rarely wiped out, their inclusion in the analysis produces spuriously high levels of concordance.) Variables were ranked with respect to edge-weights after successive sets of 10 restarts, and these 10 sets of ranks were used to calculate W. It was suggested earlier that using dom/wtdg with restart may have the untoward result of allowing the critical variables to fall back down the ordering. The results of the test of concordance are consistent with this hypothesis. The concordance of the top 20 variables ranked by edge-weight was always 0.65 or less for dom/wtdg variable selection, while for random variable selection the concordance of the top 20 variables was always 0.85 or greater.

This difference can also be shown graphically. Figure 2 shows the weights of the same top 20 ranked variables of Figure 1, after 20 restarts for a random probing run and from a repeated weighted-degree. Note how the repeated weighted-degree slope is nearly horizontal, while for random probing the same first few variables are already in order.

These results show that restart with random variable ordering results in more stable patterns of edge-weights among variables across restarts. This, in turn, implies that variables that are truly 'critical', i.e. those associated with global difficulties in the sense used above, can be found early in the sequence of repeated runs with this strategy of probing for

failure.

Discussion

Domain/weighted-degree is a powerful heuristic that already demonstrates the benefits of learning from failure. Nonetheless, we have shown that further information can be extracted and utilised to improve search even more. Due to the expense entailed by information gathering, the overall efficiency of our best procedure is not appreciably different from search using the original heuristic for the class of problems tested. However, the potential for improvement exists and we do not yet know what the limit of this improvement actually is.

A critical feature of our procedure is restarting after a limited amount of search. However, in our work we have shown that this technique cannot simply be combined with the process of incrementing edge-weights in order to be successful. As discussed above, restarting under these conditions can actually 'distort' a good weight profile, evidently by biasing failure in favor of variables that are instantiated later in search and by altering this bias from restart to restart so that differences among variables are obscured.

An important result found in connection with the random variable selection procedure was that number of restarts is a more important parameter than cutoff-level. In fact, differences associated with the latter were not statistically significant over the range of values tested (Table 6). This conclusion was supported by the post-hoc analysis, which showed that, for conditions in which the maximum allotment of nodes was the same, a greater number of restarts was more beneficial than a higher cutoff.

In further experimental tests, we obtained evidence that supports our explanation of the deficiencies of weighted-degree with restart and also helps explain the improvement observed when restart is combined with random variable selection. Thus, we have shown that when variables are ranked in terms of edge-weights, the rankings of individual variables are much less stable across restarts when selection is by dom/wtdg with accumulated weights than they are when selection is random. And we have shown that differences in edge-weights are much greater when the latter procedure is used, which should translate into more effective discrimination during variable selection.

As in most learning from search approaches, there are important blame assignment issues (Sleeman, Langley, & Mitchell 1982). First, one doesn't know the actual root of any insoluble subtree without knowing the solution, because search can backtrack above any given variable except, of course, the first variable selected. Secondly, when propagating assignments using arc consistency, one doesn't know that a given constraint is truly to blame for a variable's domain going empty. It could be that the constraint of a neighboring variable had removed so many values that support was lost for neighboring domains and thus it should be the former which is weighted. In this context, a significant advantage of the weighted degree heuristic is that the weight it gives for each individual wipeout is small. In this case, it is the cumulative effect of a variable's constraints directly causing domain wipeouts on a number of occasions that determines whether that variable will be selected over another.

It should be noted that, in these procedures, the extra work done before solving is simply heuristic search using either weighted-degree or random variable ordering. The overall time taken per problem should be similar to the weighted-degree heuristic without restarts, if the number of nodes required to solve the problem is the same as the total number of nodes required by our method.

Related Work

Constraint weighting was introduced by Morris (Morris 1993) in his breakout algorithm for escaping local minima while using local search, and a similar technique was introduced by Selman (Selman & Kautz 1993) for solving large structured SAT problems. Recently Eisenberg and Faltings (Eisenberg & Faltings 2003) suggested using Morris's breakout algorithm to identify hard and unsolvable subproblems. After a fixed number of iterations through the algorithm they switched to a systematic search technique for solving the problem which uses a static variable ordering based on constraint weights produced by the breakout algorithm and the graph structure. Their method differs from ours in its strategy for knowledge acquisition: they used local search techniques whilst we use probes of the search space. Furthermore, our goal is purely to boost the power of the weighted-degree heuristic by supplying it with information for early decisions.

Squeaky Wheel Optimization (SWO) (Joslin & Clements 1998) is another local search strategy that assigns blame to trouble elements in the form of numeric values and then sorts the ordering for the next attempt at solving the problem based on these values. However, unlike our method, SWO does not carry blame factors over from run to run and does not have a completeness guarantee.

Chaff, as mentioned earlier, is a SAT solver (Moskewicz, Madigan, & Malik 2001) which uses restarts with clause learning for effective problem solving. Due to the addition of conflict clauses through learning, SAT ensures that it does not repeat the previous attempt when restarting. This is similar to using restarting with weighted-degree in our case, where incrementing weight counters ensures that the previous search is not repeated upon restart. However they use a weight-decaying method that gives recent weight updates more bearing. This is because they want to concentrate on penalties which are more relevant to the current search space. We, on the other hand, are trying to identify constraints related to global difficulties in order to make the best early selections for our final run.

(Tompkins & Hoos 2004) presented experimental evidence indicating that local search algorithms with clause-weight learning do not benefit from their weights on a global scale, and that all learning is local. Our findings with the repeated weighted-degree method were similar in that restarting with the learned weights proved detrimental. However, we would argue that different methods give information of different quality, and it is the method which is key. We have shown that when we use random probing of the search space to find constraint weights, these have a global benefit for the

weighted-degree heuristic. It should be noted though that our work was for complete search for CSPs whilst theirs was for local search with SAT, which could also have had a bearing.

Refalo's work on impact-based search strategies used a similar probing of the search space with restarts to uncover effects of search specific to a problem (Refalo 2004). In this approach he measured the "impact" (reduction of the search space due to assignment of a variable) of values for variables and thus of the variables themselves by the domain reduction caused by the propagation of that value. He found that using these probes for information gathering prior to solving boosted search. Although the information gathered and the use of it is different, the idea of probing the search space is similar to ours.

Conclusions and Future Work

In devising learning strategies for problem solving there are some key questions that one must consider: what are we going to learn, how are we going to learn it, how can we use the information learnt and most importantly how dependable is the information.

In our case, the information learnt was the number of times the constraints of each variable directly caused a domain wipeout. Learning was encoded by incrementing the weight on each edge, and this information was then used to guide search by selecting variables in decreasing order of their edge-weight sum. Since the dependability of the information in singularity is quite weak, weights are only incremented by 1. Thus, it is only through the buildup of these weights that importance is attached to a variable.

We presented two approaches which affected the quality of information learnt during preprocessing, namely weighted-degree variable selection and random variable selection. Clearly, using the weighted-degree heuristic in this manner is not a dependable method for information gathering, although it did allow us to solve problems during preprocessing. Random variable ordering was much more successful, presumably because it probed more diverse areas of the search space for information. We showed that this preprocessing method of learning from failure can improve on the weighted-degree heuristic or a variant by supplying it with relevant information for its initial variable selections as well as later in search.

Clearly, the random-selection strategy is not suitable for solving easy problem instances, since the cost of probing the search space is likely to outweigh the benefit of the information learnt. However, we have shown that on hard problem sets, information gathering prior to solving can be beneficial.

An important question is whether we can obtain the information required to improve variable selection with less cost. One method may be to improve the quality of the information learned. We have two approaches in mind for doing this: depth restriction for preprocessing and inverse-depth weighting. Both make use of the insights of this paper, that constraints causing domain wipeouts early in search are more useful for learning than those later in search.

References

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *Proc. Sixteenth European Conference on Artificial Intelligence-ECAI'04*, 146–150.

Eisenberg, C., and Faltings, B. 2003. Using the breakout algorithm to identify hard and unsolvable subproblems. In Rossi, F., ed., *Principles and Practice of Constraint Programming-CP'03. LNCS No. 2833*, 822–826.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proc. Tenth European Conference on Artificial Intelligence-ECAI'92*, 31–35.

Haralick, R. M., and Elliott, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–314.

Hays, W. L. 1973. *Statistics for the Social Sciences*. Holt, Rinehart, Winston, 2nd edition.

Hulubei, T., and O'Sullivan, B. 2006. The impact of search heuristics on heavy-tailed behaviour. *Constraints Journal* 11:157–176.

Joslin, D., and Clements, D. 1998. Squeaky wheel optimization. In *Proc. Sixteenth National Conference on Artificial Intelligence-AAAI'98*, 340–346.

Katsirelos, G., and Bacchus, F. 2005. Generalized nogoods in CSPs. In *Proc. Twentieth National Conference on Artificial Intelligence-AAAI'05*, 390–396.

Lecoutre, C.; Boussemart, F.; and Hemery, F. 2004. Backjump-based techniques versus conflict-directed heuristics. In *Sixteenth International Conference on Tools with Artificial Intelligence-ICTAI'04*, 549–557.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proc. Eleventh National Conference on Artificial Intelligence-AAAI'93*, 40–45.

Moskewicz, M.; Madigan, C.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proc. Design Automation Conference*, 530–535.

Refalo, P. 2004. Impact-based search strategies for constraint programming. In Wallace, M., ed., *Principles and Practice of Constraint Programming-CP'04. LNCS No. 3258*, 557–571.

Selman, B., and Kautz, H. 1993. Domain-independent extensions to GSAT : solving large structured satisfiability problems. In *Proceedings of IJCAI '93*, 290–295.

Sleeman, D.; Langley, P.; and Mitchell, T. M. 1982. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine* 3:48–52.

Tompkins, D. A. D., and Hoos, H. H. 2004. Warped landscapes and random acts of SAT solving. In *Proc. Eight International Symposium on Artificial Intelligence and Mathematics-ISAIM04*.