Search Ordering Heuristics for Restarts-based Constraint Solving

Margarita Razgon and **Barry O'Sullivan** and **Gregory M. Provan**
Department of Computer Science, University College Cork, Ireland
{m.razgon|b.osullivan|g.provan}@cs.ucc.ie

Abstract

Constraint Satisfaction Problems are ubiquitous in Artificial Intelligence. Over the past decade significant advances have been made in terms of the size of problem instance that can be solved due to insights gained from the study of runtime distributions of systematic backtrack search algorithms. A particularly impressive advance has been the use of randomization and restarts, which are now a standard component of state-of-the-art solvers. In this paper we propose a new class of variable and value ordering heuristics that learn from nogoods without storing them. The empirical analysis provides clear evidence that the proposed ordering heuristics dramatically improve the performance of restarts-based constraint solving. We can regard our heuristics as exploiting *positive experience* to improve search.

Introduction

There are essentially two approaches to solving Constraint Satisfaction Problems: backtrack search and stochastic local search. The advantage of the former is that it is complete, while the latter, while incomplete, can scale to very large problem instances. Restarts-based algorithms combine the advantages of both these approaches. The idea of restarts is based on the recently discovered phenomenon that the runtime distributions for many classes of soluble problems are heavy-tailed distributions (Gomes *et al.* 2000).

Inspired by heavy-tailed distributions, randomization and rapid random restarts have proven to be extremely useful for solving very large problem instances. When we use a rapid random restarts approach to solving, we essentially try to find one of the very short runs that contribute to the low median runtime that characterize heavy-tailed runtime distributions. By setting a restart cutoff, we define an upper bound on the number of backtracks in any run. If this cutoff is exceeded, we simply abandon the current search and restart. By ensuring there is a degree of randomness in our search heuristics, we can be confident that we will search for a solution in quite a different way, hopefully solving the problem quickly by finding one of those short runs.

Rapid random restarts have been shown to exploit the fact that many real world problem instances have small backdoor sets of variables (Williams, Gomes, & Selman 2003). If one assigns the variables in a backdoor set correctly, we can solve the remainder of the instance in polynomial time. If the backdoor set of variables is small enough, and we set an appropriate restart cutoff, we can provably solve the problem in time exponential in the backdoor size. For backdoors that are of logarithmic size, this implies we can solve problems in polynomial time given a backdoor set of variables. The restarts approach has been successfully applied to a number of search problems (Gomes *et al.* 2000; Moskewicz *et al.* 2001; Gomes *et al.* 1998), with problems involving millions of variables being within the reach of systematic solvers.

When we solve a problem instance using a restarts-based approach, we have an opportunity to learn from each run that must be abandoned before we restart. For example, any failures we encountered in previous runs will also constitute failures in any subsequent run. In this paper we present a novel approach to learning from such failures.

In this paper we present novel variable and value ordering heuristics based on learning in a restarts context. The learning component of the heuristics maintains a counter for each assignment of a value v to a variable, which reflects the number of times v appears in a partial solution. The counters are updated each time we backtrack during search: the counters associated with the assignments of the discarded partial solution are increased by one. Our proposed *variable ordering heuristic* selects the variable with the *smallest sum of counters* of values in its current domain. The intuition behind this heuristic is that it increases the level of *exploration* performed by search when restarting. The *value ordering heuristic* is based on the assumption that a value appearing frequently in many partial solutions is more likely to appear in a full solution. Therefore, the heuristic selects, for the chosen variable, a value with the *largest valued counter*. This is similar, but not equivalent, to the traditional *minimize inconsistency* value ordering used in backtrack search (Frost & Dechter 1995).

To evaluate the proposed heuristics, we performed experiments on a variety of problem classes. The experiments provide clear evidence that our heuristics significantly improve the performance of restarts, when compared to a standard baseline algorithm.

The remainder of the paper is organized as follows. We first present some essential background and terminology re-

quired for this paper. We then describe our new variable and value ordering heuristics. We report on experiments performed on Quasigroup Completion Problems, Langford's Number Problem and Random Graph k-Colouring Problem, showing very positive results. A summary of the most relevant related work is then presented. Finally, we draw our conclusions and present a number of potential extensions to our work.

Background

A *Constraint Satisfaction Problem* (CSP) (Dechter 2003) is a 3-tuple $Z \hat{=} \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is a finite set of variables $\mathcal{X} \hat{=} \{x_1, \ldots, x_n\}$, $\mathcal{D}$ is a set of finite domains $\mathcal{D} \hat{=} \{D(x_1), \ldots, D(x_n)\}$ where the domain $D(x_i)$ is the finite set of values that variable x_i can take, and a set of constraints $\mathcal{C} \hat{=} \{c_1, \ldots, c_m\}$. Each constraint c_i is defined by the ordered set $var(c_i)$ of the variables it involves, and a set $sol(c_i)$ of allowed combinations of values. An *assignment* of Z is a pair (x_i, v_i) such that $x_i \in \mathcal{X}$ and $v_i \in D_{x_i}$. A *set of assignments* to the variables in $var(c_i)$ *satisfies* c_i if it belongs to $sol(c_i)$. A set of assignments is a *partial solution* of Z if it satisfies any constraint in $\mathcal{C}$. A partial solution that assigns all the variables of Z is a *solution* of Z. A CSP that has no solution is *insoluble*. Generally, not every partial solution can be extended to a full solution. If a partial solution is not a subset of any solution, it is called a *nogood* (Schiex & Verfaillie 1994; Dechter 1990).

In this paper we use Forward Checking (FC) (Haralick & Elliott 1980) as the underlying constraint solver. FC is a backtrack search-based solver that ensures that whenever a variable x_i is assigned, the values incompatible with the *current* partial solution are discarded from the domains of all the unassigned variables constrained by x_i. The values of x_i that are not discarded are called *feasible* and the set of all feasible values of x_i is called the *current domain* of x_i. If, as a result of this domain filtering the current domain of a variable becomes empty, backtracking is initiated immediately.

Heuristics for Restarts-based Solving

As discussed in the introduction, we can use the notion of restarts to solve a problem instance using fixed-length short runs of a randomized backtrack (here FC) procedure. If the number of backtracks in the current run of FC is greater than the given cutoff but the given CSP is not solved so far, the current run stops and the new run starts. The procedure finishes when FC solves the given CSP using a number of backtracks that is less than the given cutoff. If the given CSP is not solved during a maximum number of runs, we report failure.

The heavy-tailed phenomenon, demonstrated in (Gomes *et al.* 2000), suggests that restarts use computational resources more effectively than a single long run. In other words, restarts can help us find a full solution using fewer backtracks than required by a single long run. The simple intuition behind this claim is that the probability of using a sequence of assignments that lead to a *big* search tree is *reduced* by restarting with a different sequence of assignments, while the probability of finding a sequence of assignments that lead to a *small* search tree is *increased* if such sequence exists (Shlyakhter 2003). During restarts we can accumulate information about the set of nogoods of the CSP that can be useful to inform the choice of variable to branch on next and the value to assign to that variable.

In constraint satisfaction there are two guiding principles that characterize good variable and value ordering heuristics in systematic search. Firstly, when *choosing a variable*, we should follow the *fail-first* principle: choose the variable that is most like to cause us to fail (Haralick & Elliott 1980; Geelen 1992). This justification here is that if we make a mistake and move away from the path to a solution, we would like to discover this as soon as possible. Secondly, when *choosing a value for the current variable* we would like to choose one that is most *promising*, i.e. most likely to lead to a solution (Geelen 1992; Frost & Dechter 1995).

The traditional requirement when using a restarts-based search is to ensure that the choice of variable and value is randomized. At the very least we should have a variable ordering strategy that allows randomization. There are a variety of ways in which we can ensure that we randomize our heuristics. Firstly, we can simply choose a random variable or value, but we can also simply randomize a heuristic by regarding a proportion of the most highly ranked choices as being of equal quality, and selecting randomly amongst them. Our heuristics try to boost such randomization by ranking variables more highly if they have been selected less frequently in the past. In this way we ensure that the choice of variable maximizes exploration of the search tree in an attempt to find a short run to a solution. Randomization can be used to select from amongst those variables that are regarded as being in the same equivalence class in terms of quality.

The basis for our search heuristics is the cumulative number of times a variable was assigned in any run of the solver on the current problem instance. To ensure we can extract a value ordering, we associate counters with each domain value of each variable. Specifically, we record the number of times each value for each variable was assigned in a nogood of the problem. Since the number of nogoods is exponential in their length, values that occur frequently can be regarded as being highly consistent, i.e. they are consistent with many values in the domains of their neighbouring variables in the constraint graph. Therefore, high counters associated with a domain value are *positive information*. By selecting variables whose domain values have participated in few nogoods, we are picking a variable that ensures that the constraint solver explores the search space as much as possible. We discuss our variable and value ordering heuristics in greater detail below, beginning with the value ordering.

The Value Ordering Heuristic

Our choice of value is based on the assumption that an assignment that occurs in many partial solutions is likely to appear in a full solution. Based on this assumption, we propose a value ordering heuristic that assigns the given variable with a value that occurred *most frequently* in nogoods.

To implement the heuristic, we record for each variable x_i, for each value $v_i \in D(x_i)$, the number of times the assignment (x_i, v_i) appeared in a nogood. Rather than remember the nogoods we have encountered, which might exponentially increase the space and time complexity, the algorithm maintains an array of counters A with cells corresponding to all possible assignments; therefore, A stores $\mathcal{O}(nd)$ integers, where n is the number of variables and d is the maximum domain size. Initially, all the elements of A are initialized to zero. Whenever the search algorithm is about to backtrack, for each assignment (x_i, v_i) in the current partial solution, $A[(x_i, v_i)]$ is incremented by 1.

The heuristic can be invoked simply. Given a variable x_i, this variable is assigned a *feasible* value $v_i \in D(x_i)$ given by $A[(x_i, v_i)] \geq A[(x_i, v_i')]$, i.e., we assign the v_i that appeared *most often* in a nogood. If the variable x_i has never participated in a nogood, it is *randomly* assigned a feasible value $v_i \in D(x_i)$.

The Variable Ordering Heuristic

We also propose to use the information provided by nogoods to guide variable ordering. In particular, we select a variable that appears *least frequently* in nogoods. The intuition behind this variable ordering heuristic is that we boost the exploration of the search space that benefits restarts-based solving.

We implement the variable ordering heuristic using the array A described in the previous section. The proposed heuristic selects *an unassigned variable with the smallest sum of any counter of feasible values*.

To ensure that a different search tree is followed after each restart, some randomness is introduced into this variable ordering heuristic, as suggested in (Gomes, Selman, & Kautz 1998). In particular, the variables are sorted by increasing order of the sum of counters of their feasible values. The heuristic returns a variable randomly chosen among the first $H\%$ of variables of the resulting sequence (we chose $H = 25\%$).

Experiments

We present experiments to demonstrate the utility of the ordering heuristics we proposed in the previous section. We employ a constraint solver based on FC. We compare four different settings:

(1) R-WL. The baseline solver uses restarts without any learning. In this setting variables and values are selected randomly.

(2) R-VAR. We combine restarts with our variable ordering heuristic based on learning from nogoods. In this setting values are selected randomly.

(3) R-VAL. We combine restarts with our value ordering heuristic based on learning of nogoods. In this setting variables are selected randomly.

(4) R-VAR-VAL. We combine restarts with both our variable and value ordering heuristics based on learning of nogoods.

The problem domains studied, taken from CSPLIB[1], were Quasigroup Completion Problems, Langford's Number Problem and Random Graph k-Colouring Problem. In all cases the obvious binary CSP formulation was used. We measure search in terms of the number of backtracks; due to the low overhead incurred by the ordering heuristics, CPU-time was closely correlated, so we do not present it. For every tuple of parameters of a tested instance, we report results as the average of 50 instances. We set a limit of to 100 restarts for solving any instance. The proposed algorithms were implemented using Microsoft Visual C++ 6.0, and the experiments were performed under Microsoft Windows XP 2002 on a 3GHz Pentium processor using 1GB RAM.

Quasigroup Completion Problem

An order m quasigroup (Barták 2005; Gomes *et al.* 2000) is a *Latin square* of size m, that is a m by m table in which each element occurs exactly once in each row and column. An *incomplete* or *partial Latin square* is a partially filled Latin square. The *Quasigroup Completion Problem* (QCP) is the problem of determining whether the remaining entries of the incomplete Latin square can be filled in such a way that we obtain a complete Latin square.

The results of comparing the four methods for Quasigroup Completion Problem are given in Figure 1. We considered the 7×7 incomplete Latin square (a maximum of 49 variables), varying the number of pre-assigned (fixed) cells from 8 to 34. These problems were chosen so that FC could solve them in a reasonable amount of time. The pre-assigned cells that were selected randomly are presented along the horizontal axis. The chosen cutoff for restarts was 5000 backtracks. The actual numbers of backtracks required to solve the instances are presented on the vertical axis.

It is clear that the **R-VAL** (3) method *outperforms* the baseline **R-WL** (1) method in the majority of cases. We achieve up to 3 times reduction in the number of backtracks by using a value ordering heuristic in restarts **R-VAL** (3) compared with **R-WL** (1).

The method **R-VAR** (2) *outperforms* both **R-WL** (1) and **R-VAL** (3) methods for every class of QCP we considered. Using a variable ordering heuristic with restarts **R-VAR** (2) considerably reduces the number of backtracks by almost a factor of 10 in some cases over the baseline **R-WL** (1) method.

Combining both our variable and value ordering heuristics with restarts (**R-VAR-VAL** (4) method) *outperforms* all other settings methods. In particular, we achieve up to 20 times reduction in search effort compared with the baseline **R-WL** (1) method.

One can observe from the empirical results that the simple form of learning that underlies our ordering heuristics can significantly improve the performance of restarts. The improvement can be achieved by exploring the most "successful" assignments or by assigning the most "disregarded" variables. These two approaches are complementary to each other: being combined, they work better than each one separately. Furthermore, we have observed that **R-VAR-VAL**

[1] http://www.csplib.org.

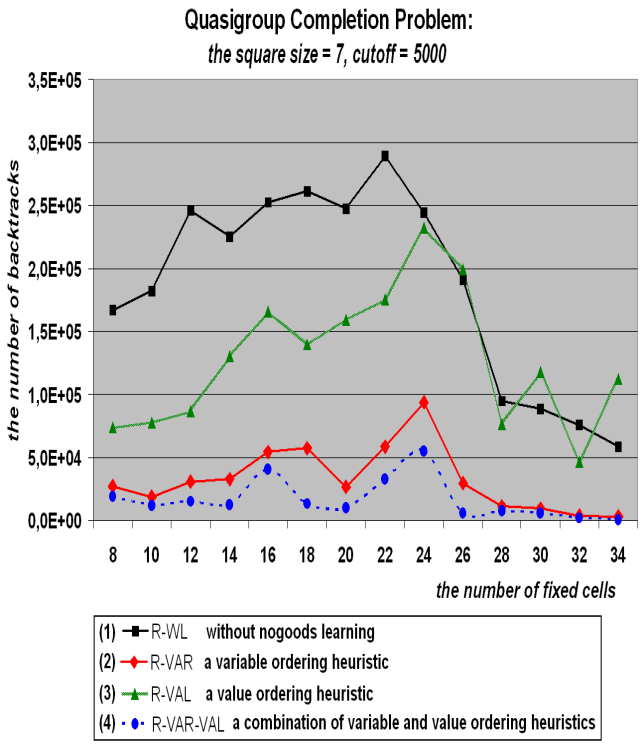

Figure 1: Experiments for Quasigroup Completion Problem.

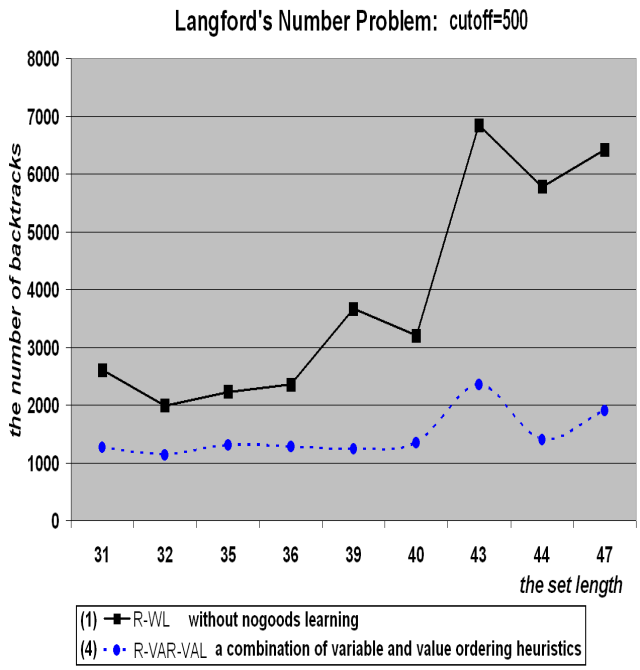

Figure 2: Experiments for Langford's Number Problem.

(4) method outperforms other heuristics on single runs, not only on average. This phenomenon evidences robustness or predictability in performance over the problems instances studied, which is a particularly attractive characteristic.

Langford's Number Problem

The task of *Langford's Number Problem* is to arrange two given sets of integers from 1 to n into single $2n$-length sequence, so that each appearance of the number m is m numbers apart from the last, for example, the two 1's appear one number apart, the two 2's appear two number apart, and so on. The problem is soluble when a set length n divided by 4 gives remainder 0 or 3.

The constraints for Langford's Number Problem are organized as follows. For each number m we save the pairs of places, where the number m could appear in $2n$-length sequence, for example the pairs of number 1 are $(1, 3)$, $(2, 4)$, and so on. The pair of number i is incompatible with the pair of number j, if these pairs contains the same place, because it is impossible for different numbers to appear on the same place.

The results of comparison of baseline **R-WL** (1) and **R-VAR-VAL** (4) methods for Langford's Number Problem are given in Figure 2 (we have chosen two the least and the most efficient methods for experiments).

The parameter of the problem is length n of two given sets of integers presented along the horizontal axis. We tested Langford's Number Problem on soluble instances. The restart cutoff was set at 500, and the actual numbers of backtracks are presented along the vertical axis.

Again, it is clear that the **R-VAR-VAL** (4) method *outperforms* the baseline method **R-WL** (1) over all sets. The savings in search effort are smaller in this case than for QCP, but the improvements are nonetheless clear (we achieve up to 4 times reduction in search effort compared with **R-WL** (1)).

Random Graph k-Colouring Problem

The task of *Graph k-Colouring Problem* is to colour each vertex of a given graph $G = (V, E)$ in one of k colours such that adjacent vertices receive different colours. We generated random Graph k-Colouring Problem by specifying the fixed numbers of vertices and colours, and varying the density (the probability for an edge between two given vertices). The resulting CSP has a set of variables corresponding to the set of vertices, the domain of every variable corresponds to the set of colours. The pairs of variables that correspond to the pairs of adjacent vertices are constrained by the inequality constraint.

The results of comparison of baseline **R-WL** (1) and **R-VAR-VAL** (4) methods for Random Graph k-Colouring Problem are shown in Figure 3.

The parameter of the problem is graph density, presented along the horizontal axis. The restart cutoff chosen here was 5000 backtracks, with the total number of backtracks required to solve the problem displayed on the vertical axis of the plot.

It should be noted that for a considerable range of densities, these problems are much too hard for either method. However, the **R-VAR-VAL** (4) method outperforms the **R-WL method** (1) for the majority of the non-phase-transition

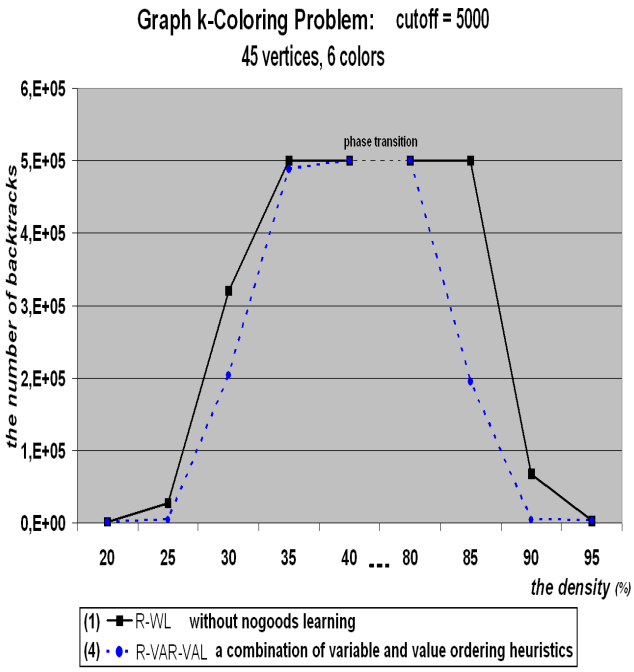

Figure 3: Experiments for Graph k-Colouring Problem.

range. In these regions we get up to a 6 times speed-up in search effort. Moreover, the range of phase-transition of **R-VAL-VAR** method is more narrow than the range of phase-transition of **R-WL** method, that is, there are points where **R-VAL-VAR** succeeds to find solution which **R-WL** fails, specifically when the graph density are 35% and 85%.

Summary of the Results

To summarize, we performed experiments on three different problems domains: Quasigroup Completion Problems, Langford's Number Problem and Random Graph k-Colouring Problem. In each case combining our variable and value ordering heuristics together (**R-VAR-VAL** (4) method) we obtained a heuristic that significantly improved the performance of restarts over a a baseline approach that chose variables and values at random (**R-WL** (1) method). In addition to improved performance, we obtained robustness in performance across a range of settings in each problem domain.

Related Work

The work presented in this paper contrasts with much of the work done on search heuristics for backtrack-based constraint solvers (Smith & Grant 1998; Geelen 1992; Haralick & Elliott 1980). These works are more concerned with characterizing what makes a good variable or value ordering decision. The predominant principle at play for variable selection is the notion of "fail-firstness", which states that "in order to succeed, try where you are most likely to fail." While recent work has cast doubt on whether such a strategy is what characterizes good search heuristics (Smith & Grant 1998), most of the heuristics currently used are, to some extent, based on this principle. In the present work we prefer to select variables whose current set of values have been tried least in the past. The benefit being that restarts-based constraint satisfaction benefits from randomization, or simply an exploration of the search space. Our variable ordering heuristic is complementary to this feature of restarts.

Our approach also contrasts with the standard approaches to handling nogoods learned during search (Schiex & Verfaillie 1994; Dechter 1990; Katsirelos & Bacchus 2003). The more traditional approaches to exploiting nogoods attempt to remember them in order to help the solver avoid repeating the same mistakes in the future, while others can modify their search ordering heuristics based on that learning (Moskewicz *et al.* 2001). Unlike these methods, the algorithm proposed in this paper learns the *number of nogoods* containing a particular value rather than the nogoods themselves. This strategy is preferable from the point of view of space complexity. According to our experiments, the proposed learning strategy also achieves significant improvements in performance over pure random restarts.

Our work also contrasts with that in (Freuder & Wallace 1995). In that work the authors generalise the context in which a nogood was found in order to leverage the experience to similar contexts. These contexts are identified by generalising nogoods using inconsistency preserving mappings. In our approach we simply use the experience of encountering nogoods to update weights on the assignments in the nogood.

A number of approaches that learn weights based on failed states in search have been reported. One of the most successful of these relates the weighted-degree family of search heuristics (Boussemart *et al.* 2004). In that work, a weight is associated with each constraint. As constraints are propagated, if the domain of a variable is wiped out, the weight of the constraint that was being filtered when the wipe-out occurred is incremented. When a variable must be selected during the search process, the weights on the constraints on that variable are used to compute its weighted-degree. This value can be substituted into all the standard variable ordering heuristics that are based on the degree of a variable. Our approach is quite different. We maintain weights for each value in each domain, and use this information to guide search in an explorative manner rather than in a fail-first manner.

Recent work has been reported that combines aspects of weight learning with restarts to improve systematic search. In (Refalo 2004) a probing technique is used to measure the impact of assignments on the domains of the remaining variables. This information is then used in a traditional systematic search setting. In (Grimes & Wallace 2006) a two-phase approach to exploiting learning in search is presented. In the first phase weights on constraints are learned in a short run, and then used in a second phase where the problem is solved without restarts. The weights that are exploited are essentially the same as those advocated in (Boussemart *et al.* 2004), but used in a restarts scenario. However, the weights that are learned seem to be difficult to use to improve search. Our approach is quite different in that we do not learn con-

straint weights, and use value-based weights to encourage exploration rather than avoid failure.

Sellmann and Ansotegui (2006) present a novel combination of local search for learning value selection in a restarts-based context. The learning scheme used in that work is different from ours, being quite coarse-grained in that they represent value orderings as partial solutions and update these only when failure is encountered. Our approach is much more dynamic, and can also drive variable selection.

Conclusions and Future Work

In this paper we applied to CSPs an algorithm combining restarts and ordering heuristics based on learning. Our experiments supported the claim that the performance of restarts can be improved by using these heuristics.

The proposed idea of learning is quite general, and can be applied to other search problems, for example to various sequencing problems like Hamiltonian Cycle, Traveling Salesman, and scheduling problems.

The work presented in this paper is the first step in a research agenda that will study the design of value weighting schemes upon which search heuristics can be easily learned and easily exploited. The key is to support online learning, i.e., we do not assume a pre-processing training phase.

We plan to extend this work by looking at different ways of updating the counters on the values that participate in a nogood. At the moment, all assignments are treated equally, by incrementing their weights by one. However, it seems reasonable to believe that the weights should also depend on the order in which the assignments were made.

Another interesting direction for future work is to extend the approach proposed here by deriving heuristics for weighted constraint problems. Weighted problems are very common in real world applications and deserve attention.

References

Barták, R. 2005. On generators of random quasigroup problems. In Hnich, B.; Carlsson, M.; Fages, F.; and Rossi, F., eds., *CSCLP*, volume 3978 of *Lecture Notes in Computer Science*, 164–178. Springer.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In de Mántaras, R. L., and Saitta, L., eds., *ECAI*, 146–150. IOS Press.

Dechter, R. 1990. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artif. Intell.* 41(3):273–312.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers.

Freuder, E. C., and Wallace, R. J. 1995. Generalizing inconsistency learning for constraint satisfaction. In *IJCAI*, 563–571.

Frost, D., and Dechter, R. 1995. Look-ahead value ordering for constraint satisfaction problems. In *IJCAI (1)*, 572–578.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In *ECAI*, 31–35.

Gomes, C. P.; Selman, B.; McAloon, K.; and Tretkoff, C. 1998. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *AIPS*, 208–213.

Gomes, C. P.; Selman, B.; Crato, N.; and Kautz, H. A. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reasoning* 24(1/2):67–100.

Gomes, C. P.; Selman, B.; and Kautz, H. A. 1998. Boosting combinatorial search through randomization. In *AAAI/IAAI*, 431–437.

Grimes, D., and Wallace, R. 2006. Learning from failure in constraint satisfaction search. In *Proceedings of the AAAI Workshop on Learning for Search*.

Haralick, R. M., and Elliott, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.* 14(3):263–313.

Katsirelos, G., and Bacchus, F. 2003. Unrestricted nogood recording in CSP search. In Rossi, F., ed., *CP*, volume 2833 of *Lecture Notes in Computer Science*, 873–877. Springer.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *DAC*, 530–535. ACM.

Refalo, P. 2004. Impact-based search strategies for constraint programming. In Wallace, M., ed., *CP*, volume 3258 of *Lecture Notes in Computer Science*, 557–571. Springer.

Schiex, T., and Verfaillie, G. 1994. Nogood Recording for Static and Dynamic Constraint Satisfaction Problem. *International Journal of Artificial Intelligence Tools* 3(2):187–207.

Sellmann, M., and Ansótegui, C. 2006. Disco - Novo - GoGo: Integrating local search and complete search with restarts. In *AAAI*. AAAI Press.

Shlyakhter, I. 2003. Main techniques for solving real-world SAT instances. *Unpublished Mansucript*.

Smith, B. M., and Grant, S. A. 1998. Trying harder to fail first. In *ECAI*, 249–253.

Williams, R.; Gomes, C. P.; and Selman, B. 2003. Backdoors to typical case complexity. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 1173–1178. Morgan Kaufmann.