

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

Over the past decade impressive advances have been made in solving Constraint Satisfaction Problems by using of randomization and restarts. In this paper we propose a new class of variable and value ordering heuristics based on learning from nogoods without storing them. We show empirically that these heuristics dramatically improve the performance of restarts-based constraint solving.

Introduction

The idea of restarts (fixed-length short runs of a randomized backtrack procedure), has been successfully applied to search problems (Gomes *et al.* 2000; 1998). When we solve a Constraint Satisfaction Problem instance using a restarts-based approach, we have an opportunity to learn from each run that must be abandoned before we restart. In this paper we present novel variable and value ordering heuristics based on learning in a restarts context.

Our approach contrasts with standard approaches to handling nogoods learned during search (Schiex & Verfaillie 1994; Katsirelos & Bacchus 2003), which remember nogoods in order to avoid the same mistakes in the future; our algorithm learns the *number of nogoods* containing a particular value rather than the nogoods themselves. This is preferable from the point of view of space and time complexity. Our value heuristics also differs from a novel combination of local search for learning value selection in a restarts-based context (Sellmann & Ansótegui 2006), which 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.

We performed experiments on a variety of problem classes taken from CSPLIB (<http://www.csplib.org>). Due to space constraints, we present results only for Quasigroup Completion Problem. All experiments provide clear evidence that our heuristics significantly improve the performance of restarts.

Background

A *Constraint Satisfaction Problem* (CSP) is a 3-tuple $Z \triangleq \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} \triangleq \{x_1, \dots, x_n\}$ is a finite set of vari-

ables, $\mathcal{D} \triangleq \{D(x_1), \dots, D(x_n)\}$ is a set of finite domains, and $\mathcal{C} \triangleq \{c_1, \dots, c_m\}$ is a set of constraints. 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 . If a partial solution is not a subset of any solution, it is called a *nogood*. 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 $D(x_i)$ that are not discarded are called *feasible* and the set of all feasible values of $D(x_i)$ is called the *current domain* of x_i . When the current domain of a variable becomes empty during this filtering, backtracking is initiated immediately.

Heuristics for Restarts-based Solving

From each run we accumulate information about nogoods of the CSP. We propose new variable and value ordering heuristics based on learning from nogoods without storing them, using the cumulative number of times a variable was assigned in any run of the solver on the current problem instance.

The Value Ordering Heuristic

We choose a variable’s value based on the assumption that an assignment that occurred in many partial solutions is likely to appear in a full solution. The proposed value heuristic assigns the given variable with a value that occurred *most frequently* in nogoods. We record for each assignment (x_i, v_i) the number of times it appeared in a nogood, and maintain an array of counters A with cells corresponding to all possible assignments. Initially, all the elements of A are 0. Whenever the search algorithm is about to backtrack, $A[(x_i, v_i)]$ is incremented by 1 for each assignment (x_i, v_i) in the current partial solution.

The heuristic is invoked simply. Given a variable x_i , it is assigned with a value $v_i \in D(x_i)$ such that the assignment

Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

(x_i, v_i) is *feasible* and for each other feasible assignment (x_i, v'_i) , $A[(x_i, v_i)] \geq A[(x_i, v'_i)]$, i.e. v_i that appeared *most often* in a nogood. If the variable x_i has never participated in a nogood, it is *randomly* assigned with a feasible value.

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 is that we boost the exploration of the search space that benefits restarts-based solving. For implementation we use the array A described in the previous section. The proposed heuristic selects *an unassigned variable with the smallest sum of any counter of its feasible values*. To ensure that a different search tree is followed after each restart, some randomness is introduced (Gomes, Selman, & Kautz 1998). 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 compare four following methods.

- (1) **R-WL.** The baseline solver uses restarts without any learning, variables and values are selected randomly.
- (2) **R-VAR.** We combine restarts with our variable ordering heuristic, when values are selected randomly.
- (3) **R-VAL.** We combine restarts with our value ordering heuristic, when variables are selected randomly.
- (4) **R-VAR-VAL.** We combine restarts with both our variable and value ordering heuristics.

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, when a limit for solving any instance is set to 100 restarts.

An order m quasigroup (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. The *Quasigroup Completion Problem* (QCP) is the problem of determining whether the remaining entries of the partially filled Latin square can be filled to obtain a complete one.

The results of comparing the four methods for QCP are given in Figure 1. We achieve up to 3 times reduction in the number of backtracks by using **R-VAL** (3) compared with **R-WL** (1) in the majority of cases. Using **R-VAR** (2) method considerably reduces the number of backtracks by almost a factor of 10 over the baseline **R-WL** (1) method. The method **R-VAR-VAL** (4) *outperforms* all other settings methods, achieving up to 20 times reduction in search effort compared with the baseline **R-WL** (1) method.

Conclusions

The best improvement of restarts-based constraint solving is achieved by the combination of exploring the most “success-

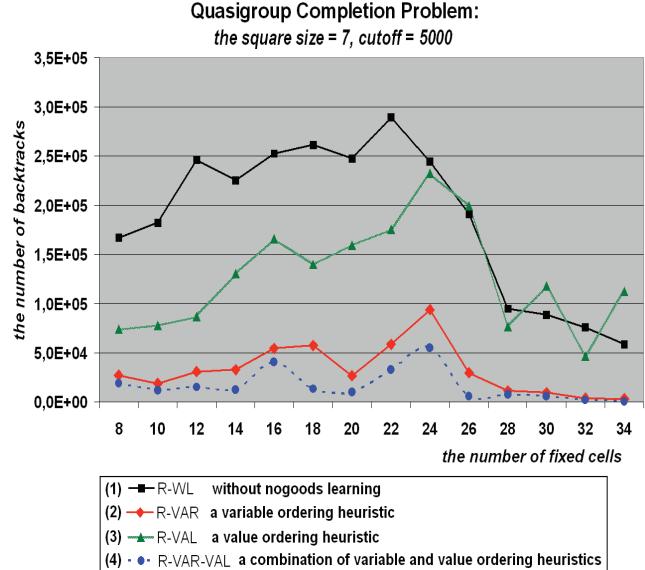


Figure 1: Experiments for Quasigroup Completion Problem.

ful” assignments and assigning the most “disregarded” variables (**R-VAR-VAL** (4) method). Furthermore, this method outperforms other heuristics on single runs, not only on average. This provided robustness in performance over the problems instances studied.

Acknowledgements: Razgon and Provan are supported by Science Foundation Ireland (Grant Number 04/IN3/I524). O’Sullivan is supported by Science Foundation Ireland (Grant Number 05/IN/I886).

References

- 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.
- 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.
- 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.