A Meta-heuristic for Subset Problems

Pierre Flener, Brahim Hnich, and Zeynep Kızıltan

Computer Science Division, Department of Information Science
Uppsala University, Box 513, S – 751 20 Uppsala, Sweden

Abstract. In constraint solvers, variable and value ordering heuristics are used to fine-tune the performance of the underlying search and propagation algorithms. However, few guidelines have been proposed for when to choose what heuristic among the wealth of existing ones. Empirical studies have established that this would be very hard, as none of these heuristics outperforms all the other ones on all instances of all problems (for an otherwise fixed solver). The best heuristic varies not only between problems, but even between different instances of the same problem. Taking heed of the popular dictum “If you can’t beat them, join them!” we devise a practical meta-heuristic that automatically chooses, at run-time, the “best” available heuristic for the instance at hand. It is applicable to an entire class of NP-complete subset problems.

1 Introduction

If you can’t beat them, join them!
— Anonymous

Constraint Satisfaction Problems (CSPs) — where appropriate values for the problem variables have to be found within their domains, subject to some constraints — represent many real life problems. Examples are production planning subject to demand and resource availability, air traffic control subject to safety protocols, transportation scheduling subject to initial and final location of the goods and the transportation vehicles, etc. Many of these problems can be expressed as constraint programs and then be solved using constraint solvers.

Constraint solvers (such as sicstus CLP(FD) [2] and opl [13]) are equipped with constraint propagation algorithms based on consistency techniques such as bounds consistency, plus a search algorithm such as forward-checking, and labelling heuristics, one of which is the default. To enhance the performance of a constraint program, a lot of research has been made in recent years to develop new heuristics concerning the choice of the next variable to branch on during the search and the choice of the value to be assigned to that variable, giving rise to variable and value ordering (VVO) heuristics. These heuristics significantly reduce the search space [10]. However, little is said about the application domain of these heuristics, so programmers find it difficult to decide when to apply a particular heuristic, and when not.
In order to understand our terminology, note that the phrase *problem class* here refers to a whole set of related problems, while the term *problem* designates a particular problem (within a class), and the word *instance* is about a particular occurrence of a problem. (We here identify problems with their chosen models.) For example, planning is a problem class, travelling salesperson is a problem within that class, and visiting all capital cities of Europe is an instance of that problem. Much of (constraint) programming research is about pushing results from the instance level to the problem level, if not to the problem-class level, so as to get generic results.

The difficulty of mapping the right heuristic to a given problem is mainly due to the following. As shown by Tsang et al. [17], there is no universally best solver for all instances of all problems. Thus, we are only told that a particular solver is “best” for the particular instances used by researchers to carry out their experiments. Therefore, as also noticed by Minton [14], the performance of solvers is instance-dependent, i.e., for a given problem a solver can perform well for some (distributions on the) instances, but very poorly on others.

In such a case, conventional wisdom suggests joining the competitors, although we propose a novel way of interpreting this popular dictum: rather than joining efforts with the competitors (by teaming up with some of them), we advocate joining the efforts of the competitors, thus “acquiring” some of them and being at the helm! But, how can this be done here, as a solver cannot know in what situation it is? The answer is to do the investigation at the level of problem classes, and to enrich the solver accordingly.

Assuming that we have a set $H$ of VVO heuristics (including the default one), we take an empirical approach to completely pre-determine a meta-heuristic that can decide which available heuristic in $H$ “best” suits the instance to be solved, and this for any instance of any problem of the considered class. (We here use constraint solvers as blackboxes, thus fixing the propagation and search algorithms.) Such a meta-heuristic can then be added to the constraint solver. We here illustrate our approach with an NP-complete class of subset problems.

This paper is organised as follows. In Section 2, we discuss a class of subset problems and show the generic finite domain constraint store that results from such problems. Then, in Section 3, we present our empirical approach and devise our meta-heuristic for subset problems. Finally, in Section 4, we conclude, compare with related work, and discuss our directions for future research.

## 2 Subset Decision Problems

We assume that CSP models are initially written in a very expressive, purely declarative, typed, set-oriented constraint programming language, such as our ESRA [6], which is designed to be higher-level than even OPL [18]. We can automatically compile ESRA programs into lower-level finite-domain constraint languages such as CLP(FD) or OPL. The purpose of this paper is not to discuss how this can be done, nor the syntax and semantics of ESRA.

---

1 This paper is an extension of the unrefered [14].