Exploiting subproblem dominance
in constraint programming

Geoffrey Chu · Maria Garcia de la Banda ·
Peter J. Stuckey

Published online: 16 November 2011
© Springer Science+Business Media, LLC 2011

Abstract Many search problems contain large amounts of redundancy in the search. In this paper we examine how to automatically exploit subproblem dominance, which arises when different partial assignments lead to subproblems that dominate (or are dominated by) other subproblems. Subproblem dominance is exploited by caching subproblems that have already been explored by the search, using keys that characterise the subproblems, and failing the search when the current subproblem is dominated by a subproblem already in the cache. In this paper we show how we can automatically and efficiently define keys for arbitrary constraint problems using constraint projection. We show how, for search problems where subproblem dominance arises, a constraint programming solver with this capability can solve these problems orders of magnitude faster than solvers without caching. The system is fully automatic, i.e., subproblem dominance is detected and exploited without any effort from the problem modeller.

Keywords Caching · Dominance · Search

G. Chu · P. J. Stuckey (✉)
National ICT Australia, Victoria Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, Melbourne, Australia
e-mail: pjs@csse.unimelb.edu.au

G. Chu
e-mail: gchu@csse.unimelb.edu.au

M. García de la Banda
Faculty of Information Technology, Monash University, Clayton, VIC, Australia
e-mail: mbanda@infotech.monash.edu.au
1 Introduction

When solving a combinatorial search problem, it is common for the search to do redundant work due to the existence of different search paths leading to subproblems whose information can be somehow re-used by other subproblems. For example, the solutions and failures to subproblem \( P \) can be re-used by subproblem \( P' \) if \( P \) and \( P' \) satisfy certain properties (e.g., if they are equivalent, if they are symmetric, or if \( P' \) is dominated by \( P \)). There are a number of different methods that avoid (or reduce) redundant search, including caching solutions (e.g. [20]), symmetry breaking (e.g. [9]), and nogood learning (e.g. [15]).

This paper focuses on caching, which works by storing information in a cache regarding every new subproblem explored during the search. Right before a subproblem is explored, the search performs a lookup on the cache to check whether it contains an already explored subproblem whose information (such as failure, solutions or a bound on the objective function) can be used for the current subproblem. If so, the search does not explore the current subproblem and, instead, uses the stored information. Otherwise, the search continues exploring the subproblem and, once this is done, it stores the appropriate subproblem information in the cache.

For caching to be useful, the lookup operation must be efficient. A popular way to implement this operation is to map each subproblem to a key which is then used to store and access the information for that subproblem. The mapping is done in such a way that any test for re-usability can be performed on the subproblem keys rather than on the subproblems themselves. In the case of subproblem equivalence (which is the most restrictive form of re-usability and, thus, simplest to detect) the subproblems are often mapped to the same key and, thus, the test is a simple equality. Other kinds of re-usability might require more complex tests, such as logical entailment.

This paper explores how to use caching automatically to avoid redundancy in constraint programming (CP) search. While caching has been previously used in CP search, it has either relied on the careful manual construction of the key for each model and search strategy (e.g. [20]), or on being able to decompose the problem into independent components by fixing some of its variables (e.g. [11, 13]). Instead, we describe an approach that can automatically detect and exploit caching opportunities in arbitrary optimization problems, and does not rely on decomposition. The principal insight of our work is to define a key that can be efficiently computed during the search and can uniquely identify a relatively general notion of re-usability that we will refer to as subproblem dominance. The key is obtained by simply projecting each primitive constraint onto the set of subproblem variables not yet fixed to a particular value. We experimentally demonstrate the effectiveness of our approach, which has been implemented in the competitive CP solver Chuffed. We also provide interesting insights into the relationships between subproblem dominance and dynamic programming, symmetry breaking and nogood learning.

The rest of the paper is organized as follows. The next section provides the necessary background for constraint programming and the notation that will be used throughout the paper. Section 3 defines subproblem dominance and subproblem equivalence, and shows how we can make use of these notions to improve search by caching. Section 4 gives some examples of manual keys to support caching of specific models with specific search strategies. Section 5 shows how we can define keys for arbitrary problems on a per constraint basis and taking into account the