Improving the Performance of maxRPC

Thanasis Balafoutis¹, Anastasia Paparrizou², Kostas Stergiou¹,²,* and Toby Walsh³

¹ Department of Information and Communication Systems Engineering, University of the Aegean, Greece
² Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece
³ NICTA, University of New South Wales, Australia

Abstract. Max Restricted Path Consistency (maxRPC) is a local consistency for binary constraints that can achieve considerably stronger pruning than arc consistency. However, existing maxRPC algorithms suffer from overheads and redundancies as they can repeatedly perform many constraint checks without triggering any value deletions. In this paper we propose techniques that can boost the performance of maxRPC algorithms. These include the combined use of two data structures to avoid many redundant constraint checks, and heuristics for the efficient ordering and execution of certain operations. Based on these, we propose two closely related maxRPC algorithms. The first one has optimal \(O(n^3d^3)\) time complexity, displays good performance when used stand-alone, but is expensive to apply during search. The second one has \(O(en^2d^4)\) time complexity, but a restricted version with \(O(en^4d)\) complexity can be very efficient when used during search. Both algorithms have \(O(ed)\) space complexity when used stand-alone. However, the first algorithm has \(O(edn)\) space complexity when used during search, while the second retains the \(O(ed)\) complexity. Experimental results demonstrate that the resulting methods constantly outperform previous algorithms for maxRPC, often by large margins, and constitute a more than viable alternative to arc consistency.

1 Introduction

maxRPC is a strong domain filtering consistency for binary constraints introduced in 1997 by Debruyne and Bessiere [5]. maxRPC achieves a stronger level of local consistency than arc consistency (AC), and in [6] it was identified, along with singleton AC (SAC), as a promising alternative to AC. Although SAC has received considerable attention since, maxRPC has been comparatively overlooked. The basic idea of maxRPC is to delete any value \(a\) of a variable \(x\) that has no arc consistency (AC) or path consistency (PC) support in a variable \(y\). A value \(b\) is an AC support for \(a\) if the two values are compatible, and it is also a PC support for \(a\) if this pair of values is path consistent. A pair of values \((a, b)\) is path consistent iff for every third variable there exists at least one value, called a PC witness, that is compatible with both \(a\) and \(b\).

The first algorithm for maxRPC was proposed in [5], and two more algorithms have been proposed since then [7,10]. The algorithms of [5] and [10] have been evaluated on

* Part of this work was carried out when the 3rd author was at NICTA, Australia.
random problems only, while the algorithm of [7] has not been experimentally evaluated at all. Despite achieving considerable pruning, existing maxRRC algorithms suffer from overhead and redundancies as they can repeatedly perform many constraint checks without triggering any value deletions. These constraint checks occur when a maxRPC algorithm searches for an AC support for a value and when, having located one, it checks if it is also a PC support by looking for PC witnesses in other variables. As a result, the use of maxRRC during search often slows down the search process considerably compared to AC, despite the savings in search tree size.

In this paper we propose techniques to improve the applicability of maxRPC by eliminating some of these redundancies while keeping a low space complexity. We also investigate approximations of maxRPC that only make slightly fewer value deletions in practice, while being significantly faster. We first demonstrate that we can avoid many redundant constraint checks and speed up the search for AC and PC supports through the careful and combined application of two data structures already used by maxRPC and AC algorithms [7,10,2,8,9]. Based on this, we propose a coarse-grained maxRPC algorithm called maxRPC3 with optimal $O(\text{end}^3)$ time complexity. This algorithm displays good performance when used stand-alone (e.g. for preprocessing), but is expensive to apply during search. We then propose another maxRPC algorithm, called maxRPC3$^{\text{erm}}$. This algorithm has $O(\text{en}^2d^4)$ time complexity, but a restricted version with $O(\text{en}^4)$ complexity can be very efficient when used during search through the use of residues. Both algorithms have $O(\text{ed})$ space complexity when used stand-alone. However, maxRPC3 has $O(\text{end})$ space complexity when used during search, while maxRPC3$^{\text{erm}}$ retains the $O(\text{ed})$ complexity.

Similar algorithmic improvements can be applied to light maxRPC (lmaxRPC), an approximation of maxRPC [10]. This achieves a lesser level of consistency compared to maxRPC but still stronger than AC, and is more cost-effective than maxRPC when used during search. Experiments confirm that lmaxRPC is indeed a considerably better option than maxRPC.

We also propose a number of heuristics that can be used to efficiently order the searches for PC supports and witnesses. Interestingly, some of the proposed heuristics not only reduce the number of constraint checks but also the number of visited nodes.

We make a detailed experimental evaluation of new and existing algorithms on various problem classes. This is the first wide experimental study of algorithms for maxRPC and its approximations on benchmark non-random problems. Results show that our methods constantly outperform existing algorithms, often by large margins. When applied during search our best method offers up to one order of magnitude reduction in constraint checks, while cpu times are improved up to four times compared to the best existing algorithm. In addition, these speed-ups enable a search algorithm that applies lmaxRPC to compete with or outperform MAC on many problems.

2 Background and Related Work

A Constraint Satisfaction Problem (CSP) is defined as a tuple $(X, D, C)$ where: $X = \{x_1, \ldots, x_n\}$ is a set of $n$ variables, $D = \{D(x_1), \ldots, D(x_n)\}$ is a set of domains, one for each variable, with maximum cardinality $d$, and $C = \{c_1, \ldots, c_e\}$ is a set of $e$