Improved Algorithms for Max-Restricted Path Consistency *

Fabrizio Grandoni and Giuseppe F. Italiano

Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata"
Via del Politecnico 1
00133 Roma, Italy
{grandoni,italiano}@disp.uniroma2.it

Abstract. A binary constraints network consists of a set of n variables, defined on domains of size at most d, and a set of e binary constraints. The binary constraint satisfaction problem consists in finding a solution for a binary constraints network, that is an instantiation of all the variables which satisfies all the constraints. A value a in the domain of variable x is inconsistent if there is no solution which assigns a to x. Many filtering techniques have been proposed to filter out inconsistent values from the domains. Most of them are based on enforcing a given kind of local consistency. One of the most important such consistencies is max-restricted path consistency. The fastest algorithm to enforce max-restricted path consistency has a $O(end^3)$ time complexity and a $O(end)$ space complexity. In this paper we present two improved algorithms for the same problem. The first still has a $O(end^3)$ time complexity, but it reduces the space usage to $O(ed)$. The second improves the time complexity to $O(end^{2.575})$, and has a $O(end^2)$ space complexity.

1 Introduction

Constraint programming is a declarative programming paradigm which allows to naturally formulate computational problems [10]. A computational problem is formulated as a *constraint satisfaction problem*, which consists in deciding whether there is an instantiation of a set of variables, defined on finite domains, which satisfies a set of constraints. Any such instantiation is a *solution* for the *constraints network*. The task of the constraint programming system is to find a solution (or alternatively all the solutions, or the "best" one). Any constraint satisfaction problem can be reduced [11] to an equivalent *binary constraint satisfaction problem*, that is a constraint satisfaction problem where each constraint involves only a pair of variables.

A value a in the domain of variable x is *inconsistent* if there is no solution which assigns a to x. Inconsistent values can be filtered out from the domains

* This work has been partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT), by the Italian Ministry of University and Research (Project "ALINWEB: Algorithmics for Internet and the Web").

without loosing any solution. Since the binary constraint satisfaction problem is NP-complete, there is no hope that all the inconsistent values can be detected in polynomial time. For this reason, local consistency properties have been studied, which allow to "quickly" remove a subset of the inconsistent values.

Some of the most important such consistencies are *arc consistency* [1,9], *path inverse consistency* [3,6,7] and ℓ *inverse consistency* [6,7].

In [4], a new and promising local consistency property has been proposed: the *max-restricted path consistency*. Computational experiments give evidence that max-restricted path consistency offers a particularly good compromise between computational cost and pruning efficiency [5]. Debruyne and Bessiere [4] developed the fastest known filtering algorithm based on max-restricted path consistency, denoted by `max-RPC-1`, which has a $O(end^3)$ time complexity and a $O(end)$ space complexity, where n is the number of variables, e is the number of binary constraints and d is the size of the largest domains.

In this paper we present a new algorithm for the same task, which we denote by `max-RPC-2`, with the same time complexity as `max-RPC-1`, but with a smaller space complexity, that is $O(ed)$.

As a second contribution of this paper, we shortly describe a variant of `max-RPC-2` of $O(end^{2.575})$ time complexity and $O(end^2)$ space complexity. This algorithm makes use of fast matrix multiplication.

The remainder of this paper is organized as follows. In Section 2 we introduce some preliminaries. In Section 3 we present Algorithm `max-RPC-2`. In Section 4 we shortly describe how to reduce the time complexity via fast matrix multiplication.

2 Preliminaries

A *binary constraints network* is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X} = \{x_1, x_2 \dots x_n\}$ is a set of n variables, $\mathcal{D} = \{D_1, D_2 \dots D_n\}$ is a set of n domains of cardinality at most d, and $\mathcal{C} = \{C_{\{i_1,j_1\}}, C_{\{i_2,j_2\}} \dots C_{\{i_e,j_e\}}\}$ is a set of e binary constraints. Variable x_i is defined over domain D_i. By a_i we denote an element of D_i. For simplicity and without lost of generality, we assume that all the values in the domains are distinct. We moreover assume that the domains are ordered (the order can be fixed arbitrarily). A *value assignment* is a pair (x_i, a_i), whose meaning is that we assign the value a_i to variable x_i. An *instantiation* is a set of value assignments, one for each variable. A *binary constraint* $C_{\{i,j\}}$ describes which assignments of values to the variables x_i and x_j are mutually compatible. The constraint $C_{\{i,j\}}$ can be represented extensively through a 0-1 matrix $A_{i,j}$ which we interpret in the following way: $A_{i,j}[a_i, a_j] = 1$ if and only if the pair of value assignments (x_i, a_i) and (x_j, a_j) satisfy the constraint $C_{\{i,j\}}$. Notice that there may be pairs of variables x_i and x_j for which there is no constraint $C_{\{i,j\}}$ in $\mathcal{C}$. In that case all the assignments of values to x_i and x_j are mutually compatible. A *solution* is an instantiation which satisfies all the constraints. The *binary constraint satisfaction problem* consists in deciding whether a binary constraints network admits a solution.

A value a_i is *inconsistent* if there is no solution which assigns a_i to variable x_i. A value a_i is *max-restricted path consistent* if it has a *path-consistent support* on each variable x_j such that $C_{\{i,j\}} \in \mathcal{C}$. A path-consistent support for a_i on x_j is a value $a_j \in D_j$ such that $A_{i,j}[a_i, a_j] = 1$ and the pair $\{a_i, a_j\}$ has at least one *witness* on each variable x_k such that $C_{\{i,k\}}$ and $C_{\{j,k\}}$ belong to $\mathcal{C}$. A witness for $\{a_i, a_j\}$ on x_k is a value $a_k \in D_k$ such that $A_{i,k}[a_i, a_k] = A_{j,k}[a_j, a_k] = 1$.

A *subdomain* $\mathcal{D}'$ of $\mathcal{D}$ is a set $\{D'_1, D'_2 \dots D'_n\}$ such that, for any $i \in \{1, 2 \dots n\}$, $D'_i \subseteq D_i$. The *order* of $\mathcal{D}'$ is the sum of the cardinalities of the domains D'_i. A subdomain $\mathcal{D}'$ is max-restricted path consistent if, for any $i \in \{1, 2 \dots n\}$, D'_i is non-empty and all the values $a'_i \in D'_i$ are max-restricted path consistent. By $\mathcal{D}_{\text{max-RPC}}$ we denote the max-restricted path consistent subdomain of $\mathcal{D}$ of maximum order (if any).

3 Max-Restricted Path Consistency in Less Space

The fastest known filtering algorithm based on max-restricted path consistency, `max-RPC-1` [4], has a O(end^3) time complexity and a O(end) space complexity. In this section we present a new algorithm for the same problem, which we denote by `max-RPC-2`, with the same time complexity as `max-RPC-1` but with a smaller space complexity, that is O(ed).

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a binary constraints network. Our algorithm removes non-max-restricted path consistent values from the domains one by one, until a domain becomes empty or all the remaining values are max-restricted path consistent.

The algorithm uses two kinds of data structures. A set `DelSet` of integers and a set S_a of values for each value a. The set `DelSet` is used to keep trace of the domains from which a value has been removed. Whenever we remove a value a_i, we store i in `DelSet`. The set S_{a_i} is used to store the last path-consistent support a_j found for a_i on variable x_j, for any x_j such that $C_{\{i,j\}} \in \mathcal{C}$.

The algorithm consists of two main steps: an initialization step and a propagation step. In the initialization step we consider each value a_i and we check if it is max-restricted path consistent. If not, we remove a_i from D_i and we add i to `DelSet`. Otherwise we store the path-consistent supports found for a_i in S_{a_i}.

In the propagation step we have to propagate efficiently the effects of deletions. In fact, the deletion of one value can induce as a side effect the deletion of other values (which were previously recognized as max-restricted path consistent). There are substantially two kinds of such situations. The first case is when we delete the unique path-consistent support a_j for the value a_i on the variable x_j. The second is when we remove the unique witness a_j on variable x_j for the pair $\{a_i, a_k\}$, where a_k is the unique path-consistent support for a_i on variable x_k. In both cases the value a_i is not max-restricted path consistent.

Thus in the propagation step, until `DelSet` is not empty, we extract an integer j from `DelSet` and we proceed as follows. We consider any value a_i, with $C_{\{i,j\}} \in \mathcal{C}$, and we check if a_i is not max-restricted path consistent because of one of the

two situations described above. In particular, we first check if the last path-consistent support a_j found for a_i on x_j (which is stored in S_{a_i}), still belongs to D_j. If not, we search for a new path-consistent support for a_i on x_j, and we update S_{a_i} accordingly. If such path-consistent support does not exist, a_i is removed and i is added to `DelSet`. Then, if a_i has not been removed in the previous step, for any variable x_k such that $C_{\{i,k\}}$ and $C_{\{j,k\}}$ belong to $\mathcal{C}$, we consider the last path-consistent support a_k found for a_i on x_k (which is stored in S_{a_i}). We check if the pair $\{a_i, a_k\}$ has a witness on variable x_j (by simply considering all the values in D_j). If not, we search for a new path-consistent support for a_i on x_k, and we update S_{a_i} accordingly. If such path-consistent support does not exist, a_i is removed and i is added to `DelSet`. In both cases, when we search for a new path-consistent support, we follow the order on the corresponding domain. This way we make sure that the same potential path-consistent support is checked at most once.

Theorem 1. *Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a binary constraints network, with n variables, defined on domains of size at most d, and e constraints. Algorithm* `max-RPC-2` *computes $\mathcal{D}_{max\text{-}RPC}$ or determines that it does not exist in $\mathrm{O}(ed)$ space and $\mathrm{O}(end^3)$ time.*

Proof (Sketch). Without loss of generality, we assume $e = \Omega(n)$. Moreover we indicate with e_i the number of domains D_j such that $C_{\{i,j\}} \in \mathcal{C}$. The set `DelSet` requires $\mathrm{O}(n)$ space. For each a_i, the set S_{a_i} takes $\mathrm{O}(e_i)$ space. Then the space complexity of `max-RPC-2` is $\mathrm{O}(n + \sum_{i=1}^{k} e_i d) = \mathrm{O}(ed)$.

The time complexity is bounded by the cost of searching for witnesses (which is required for both searching for new path-consistent supports and for checking previously detected ones). Searching for a witness costs $\mathrm{O}(d)$. Then checking a potential path-consistent support a_j for a value a_i on variable x_j costs $\mathrm{O}(e_j + e_i d) = \mathrm{O}(n + e_i d)$. Value a_i has $\mathrm{O}(d)$ potential path-consistent supports on each of the $\mathrm{O}(e_i)$ variables x_j such that $C_{\{i,j\}} \in \mathcal{C}$. No potential path-consistent support is checked more than once. Thus the total cost to search for new path-consistent supports is $\mathrm{O}\left(\sum_{i=1}^{n} e_i d^2 (n + e_i d)\right) = \mathrm{O}(end^3)$. Whenever a deletion occurs into a domain D_j, we have to search for a witness on x_j for all the pairs of values $\{a_i, a_k\}$ where a_k is the current path-consistent support for a_i on x_k, and $C_{\{i,j\}}$, $C_{\{i,k\}}$ and $C_{\{j,k\}}$ belong to $\mathcal{C}$. The number of such pairs is $\mathrm{O}(e_j^2 d)$, and we can detect them in $\mathrm{O}((e_i + e_j)d) = \mathrm{O}((n + e_j)d)$ steps. Each check costs $\mathrm{O}(d)$. Since domain D_j is interested by $\mathrm{O}(d)$ deletions, the total cost of these checks is $\mathrm{O}\left(\sum_{j=1}^{n} d^2 (nd + e_j^2 d)\right) = \mathrm{O}(end^3)$. Thus the time complexity of `max-RPC-2` is $\mathrm{O}(end^3)$. □

Notice that Algorithm `max-RPC-2` may check the same potential witness for a given pair of values more than once: since these redundant checks are relatively infrequent, they do not affect the total time complexity. Algorithm `max-RPC-1` instead, avoids redundancies by storing (in $\mathrm{O}(end)$ space) all the witnesses found: this way, the space complexity is increased without reducing asymptotically the time complexity.

4 Max-Restricted Path Consistency in Less Time

The first author [7] developed a fast path inverse consistency based filtering algorithm, based on fast matrix multiplication. A similar technique can be applied to max-restricted path consistency. In particular, given a triple of variables x_i, x_j, x_k, there is a decremental procedure to maintain the number of witnesses for each pair of the kind $\{a_i, a_j\}$ on x_k, during deletions of values in D_k, which has a $O(d^{2.376})$ initialization cost, a $O(d^{0.575})$ query cost and a $O(d^{1.575})$ amortized updating cost per deletion. Using this procedure, one can develop a variant of Algorithm `max-RPC-2`, of $O(end^{2.575})$ time complexity and $O(end^2)$ space complexity. For reasons of space, we cannot enter into details.

References

1. C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, pages 179–190, 1994.
2. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
3. R. Debruyne. A property of path inverse consistency leading to an optimal PIC algorithm. In *ECAI-00, Berlin, Germany*, pages 88–92, 2000.
4. R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *Principles and Practice of Constraint Programming*, pages 312–326, 1997.
5. R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, (14):205–230, may 2001.
6. E. C. Freuder and C. D. Elfe. Neighborhood inverse consistency preprocessing. In *AAAI/IAAI, Vol. 1*, pages 202–208, 1996.
7. F. Grandoni. Incrementally maintaining the number of l-cliques. Technical Report MPI-I-2002-1-002, Max-Planck-Institut für Informatik, Saarbrücken, 2002.
8. X. Huang and V. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.
9. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
10. K. Marriott and P. J. Stuckey. *Programming with constraints: an introduction*. MIT Press, Cambridge, MA, 1998.
11. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.