Parallel Path-Consistency Algorithms for Constraint Satisfaction* 

S. Keretho  R. Loganantharaj  V. N. Gudivada  
The Center for Advanced Computer Studies  Department of Computer Science  
University of Southwestern Louisiana  Jackson State University  
Lafayette, LA 70504  Jackson, MS 39217

1 Introduction

Many problems in artificial intelligence can be formulated as constraint satisfaction problems (CSPs) [8, 6]. Given a set of n variables, \{v_1, v_2, \ldots, v_n\}, each variable \(v_i\) associated with a domain \(D_i\) of possible values, and a set of unary and binary constraints, the problem is to find one or all possible n-tuples satisfying the constraints. Since many classes of CSPs are NP-hard, a class of consistency algorithms was invented [8, 6, 7] as heuristics to eliminate local inconsistencies that cannot participate in any global solutions. In particular, the node, arc and path-consistency algorithms detect and eliminate inconsistencies involving 1, 2 and 3 variables, respectively.

As shown in [2], arc consistency computation belongs to the class of inherently sequential problems called log-space complete for \(P\) (or \(P\)-complete), this implies that a fast parallel version, i.e. parallel polylogarithmic-time algorithm (with polynomial number of processors) is unlikely to exist. However, parallel algorithms and special parallel hardware for arc-consistency computations have been shown to provide substantial speedups [2]. We have proved in [4] that path consistency computations for CSPs in general and for Allen's time interval algebra [1] are also inherently sequential. Again, even though the potential for parallel treatment is limited, it is still essential to see what can be done in applying parallelism for path consistency computations.

2 Path Consistency

For each variable \(v_i\), let the unary constraint \(P_i(x)\) define the list of allowable labels \(x\) taken from the domain \(D_i\). For each pair of variables \(v_i\) and \(v_j\), let the binary constraint \(P_{ij}(x, y)\) define the list of allowable label pairs \((x, y)\), where \(x \in D_i\) and \(y \in D_j\). A CSP can be depicted by a labelled directed graph in which the variables are represented by nodes, the unary constraints are represented by loops on the nodes, and the binary constraints by labelled directed arcs. Constraint \(P_{ij}\) is usually represented by a Boolean matrix \(R_{ij}\) whose rows correspond to the possible values of \(v_i\) and whose columns correspond to the possible values of \(v_j\). A binary constraint network is path-consistent (3-consistent) iff for any \(i, j, k\) we have \(R_{ij} = R_{ij} \& R_{ik} \& R_{kj}\), where \& corresponds to element-by-element matrix intersection, and \(\cdot\) the Boolean matrix multiplication. The first path-consistency algorithm (PC-1) given by Montanari [8] has \(O(n^5)\) serial time complexity. More efficient path-consistency algorithms can be found in [6, 7, 3].

3 Parallel Path Consistency

Ladkin and Maddux [5] proposed to use a parallel \(O(\log n)\) matrix multiplication for the for loop of PC-1, where the multiplication operator and the addition operator correspond to the composite operator, \&\&, and the intersection operator, \&, respectively. Since there are \(O(n^2)\) number of labels in the network, the outer loop (repeat loop) will take at most \(O(n^2)\) steps to terminate. Therefore, this parallel algorithm runs in \(O(n^2 \log n)\) time using \(O(n^2)\) processors.

In [3], we have proposed a \(O(n^3)\) path-consistency algorithm which requires \(O(n^2)\) space, whereas other known \(O(n^3)\) time complexity algorithms need \(O(n^3)\) space. For the complexity discussion involving the size of domains as well as the number of variables, please refer to [3]. We use this algorithm as the main framework for our proposed parallel version.

In our framework, the unary constraints are handled differently from the classical CSPs. We introduce a reference variable \(v_0\) which has only one element in its domain, say \(D_0 = \{0\}\); thus, we may treat each unary constraint \(P_i(v_i)\) as a binary constraint \(P_{ib}(0, v_i)\). Our constraint graph has only labelled directed arcs without a loop on any node, that is, there is no constraint for \(R_{ii}\) where \(i = j\). Therefore, our constraint graph is arc and path consistent iff \(R_{ij} = R_{ij} \& R_{ik} \& R_{kj}\), where \(0 \leq i, j, k \leq n\) and \(i \neq j \neq k\). In the proposed scheme, we propagate a constraint on the edge \((i, j)\) to other edges in the graph. We accomplish this by selecting and deleting a 2-tuple \((i, j)\) from a queue of constrained edges (initially consisting of all edges in the graph, or all edges in which constraints are given explicitly). Then, consider all triangles in which \((i, j)\) is one of their sides. For each triangle, the constraint on the edge \((i, j)\) can be used to refine the constraints on the other two edges.

---

*This research was supported in part by the US Army Research Office, Grant Number DAAH04-90-G-0217.

---

1 A more accurate time complexity is \(O(a n^3)\), where \(a\) is the maximum size of the domains, and \(n\) is the number of variables. For simplicity, in this paper, we mention the time complexity of all algorithms only in terms of \(n\).
Whenever, the constraint on the propagated edge is refined, it will be pushed back on the queue, since it might refine other constraints in the graph as well.

The parallel version of our path-consistency algorithm is in Figure 1. We can apply the parallelism by assigning a processor to compute the propagation for each triangle. That is, lines 5–10 can be accomplished concurrently in a constant step using \( n \) processors needed is very realistic model of computation such as exclusive-read exclusive-write (EREW) PRAM, we can perform the procedure Par-PC in \( O(n^2 \log n) \). For both models of computation, the \( O(n) \) processors needed is very practical, and they are also candidates for hardware implementation. For a machine with a small number of processors, we can easily subdivide the propagation (the for loop) that are propagated simultaneously by the processors.

The details, analysis and further results can be found in [4].

References