Parallel Path-Consistency Algorithms for Constraint Satisfaction

Peter B. Ladkin[1], Roger D. Maddux[2]

TR-89-045

August, 1989

Abstract

This paper concerns heuristic algorithms used for solution of Boolean ConstraintSatisfaction Problems, or CSPs *[Mon74, Mac77, Fre78, Mac87]*. CSPs occur particularly in areas of artificial intelligence such as vision, temporal reasoning, and truth-maintenance systems. The most common form involves binary constraints and we consider properties of binary CSPs only (we shall omit the adjective from now on). CSPs may be represented by labelled digraphs called binary constraint networks, or BCNs. Many constraint satisfaction techniques operate upon BCNs. An important property of BCNs is that of *path-consistency*, which is used extensively as a heuristic for solving CSPs (many classes of CSPs are NP-hard, e.g. *[VilKau86]*). Every BCN has a *path-consistent reduction*, and it is known that algorithms for computing it are serial $O(n^3)$ in the number of variables *[Mac77, Fre78, All83, MacFre85, MohHen86]*.

We have formulated CSPs and path-consistency computations in the framework of Tarski's relation algebra, and give a brief overview below *[Tar41, LadMad88.2]*. We give a parallel $O(n^2 \log n)$ algorithm for achieving path-consistency. We also give a class of hard examples on which all algorihtms proposed so far, and possible parallelisations of them, take time $\theta(n^2)$. This effectively constrains parallel path-consistency algorithms of the most common form (which we glorify with the name of *reduction-type*) within a fairly narrow asymptotic range.

In the next section, we introduce the relation-algebraic formulation of CSPs. We formulate some algorithms in the following section, ending with the $O(n^2 \log n)$ parallel path-consistency algorithm. In the final section, we describe the class of problems on which the reduction-type algorithms take $\theta(n^2)$ time.

---

1. International Computer Science Institute, Berkeley, California and Kestrel Institute, Palo Alto, California. Partially supported by DARPA contract N00039-88-C-0099 to Kestrel Institute, administered by the U.S. Navy. The views and conclusions expressed in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Kestrel Institute, or any agency of the United States Government.

2. Dept. of Mathematics, Iowa State University, Ames, Iowa.

Parallel Path-Consistency Algorithms for Constraint Satisfaction

Peter B. Ladkin*
Kestrel Institute
3260 Page Mill Road
Palo Alto, Ca 94304
and
International Computer
Science Institute
1947 Center St.
Berkeley, CA 94704

Roger D. Maddux
Department of Mathematics
Iowa State University
Ames, Iowa 50011

Extended Abstract

1 Synopsis

This paper concerns heuristic algorithms used for solution of Boolean Constraint Satisfaction Problems, or CSPs *[Mon74, Mac77, Fre78, Mac87]*. CSPs occur particularly in areas of artificial intelligence such as vision, temporal reasoning, and truth-maintenance systems. The most common form involves binary constraints and we consider properties of binary CSPs only (we shall omit the adjective from now on). CSPs may be represented by labelled digraphs called binary constraint networks, or BCNs. Many constraint satisfaction techniques operate upon BCNs. An important property of BCNs is that of *path-consistency*, which is used extensively as a heuristic for solving CSPs (many classes of CSPs are NP-hard, e.g. *[VilKau86]*). Every BCN has a *path-consistent reduction*, and it is known that algorithms for computing it are serial $O(n^3)$ in the number of variables *[Mac77, Fre78, All83, MacFre85, MohHen86]*.

We have formulated CSPs and path-consistency computations in the framework of Tarski's relation algebra, and give a brief overview below *[Tar41, LadMad88.2]*. We give a parallel $O(n^2 \log n)$ algorithm for achieving path-consistency. We also give a class of hard examples on which all algorithms proposed so far, and possible parallelisations of them, take time $\theta(n^2)$. This effectively constrains parallel path-consistency algorithms of the most common form (which we glorify with the name of *reduction-type*) within a fairly narrow asymptotic range.

*The first author was partially supported by DARPA contract N00039-88-C-0099 to Kestrel Institute, administered by the U.S. Navy. The views and conclusions expressed in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Kestrel Institute, or any agency of the United States Government.

In the next section, we introduce the relation-algebraic formulation of CSPs. We formulate some algorithms in the following section, ending with the $O(n^2\ log\ n)$ parallel path-consistency algorithm. In the final section, we describe the class of problems on which the reduction-type algorithms take $\theta(n^2)$ time.

2 Relation Algebra and CSPs

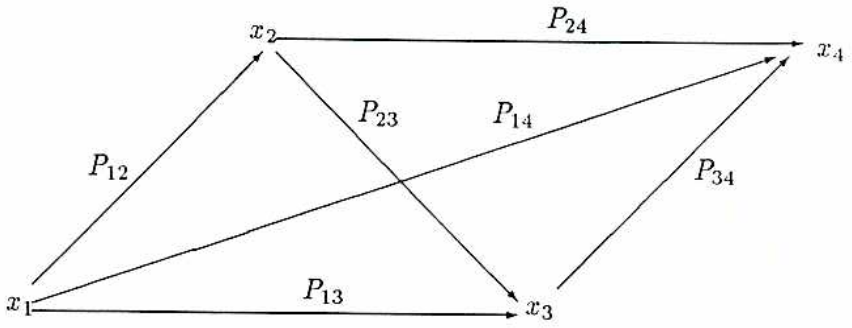
We define binary constraint satisfaction problems, their associated networks, the concept of path-consistency, and the *most general path-consistent reduction* of a network $\mathcal{A}$, $MGR_3(\mathcal{A})$. We give a general formulation of reduction-type algorithms for $MGR_3(\mathcal{A})$, and obtain the parallel $O(n^2\ log\ n)$ algorithm.

Binary Constraint Satisfaction Problems

A binary CSP, or BCSP, is given by a formula $(P_1(x_1) \wedge ...P_n(x_n) \wedge P_{12}(x_1, x_2) \wedge P_{13}(x_1, x_3) \wedge \ ..... \ \wedge \ P_{n-1,n}(x_{n-1}, x_n))$, where the P_i and P_{ij} are predicate symbols representing constraints on the variables x_i. The constraints are binary relations, i.e. sets of pairs of objects from some domain. Thus we may freely use set-theoretic operations such as union and intersection on these constraints, as we shall below, along with other operations specific to binary relations. A BCSP is *satisfied* by finding values for the variables which satisfy the formula. The unary constraints represent domain constraints on values for each of the variables, and the binary constraints are usually of the form $P_{ij}(x_i, x_j) \ \equiv \ ((x_i\ R_1\ x_j) \vee .....(x_i\ R_q\ x_j))$ where the R_p are a fixed collection of disjoint 'basic' relations over the domain of interest. For examples, and bibliography, see the survey *[Mac87]*. A BCSP may be represented as a labelled digraph, called a *binary constraint network* or BCN, where the nodes represent the variables, and the labels on the edges represent the constraint between the variables at the head and tail. Unary constraints are represented as edges with the same head and tail. See Figure 1 for an example of a BCN (with only one edge between a given node pair). We shall always consider a BCN to be a complete graph, with the *universal* constraint labelling the edge between any pair of variables not otherwise constrained (the universal constraint is the universal relation, which contains all pairs of values from the domain. See below). These conventions are necessary for the definition and computation of path-consistent networks, as we shall see.

2.1 Regular BCSPs

A BCSP is *regular* if there is only one constraint between a given pair of variables. A regular BCSP may be represented as a BCN with a single directed edge between a given pair of nodes. See Figure 1. For any BCSP there is an easily computable regular BCSP with the same set of solutions. First we define the *converse* $R^\smile$ of a relation R by the equivalence $xR^\smile y \ \equiv_{def} \ yRx$. Suppose a nonregular BCSP has both constraints $P_{ij}(x_i, x_j)$ and $P_{ji}(x_j, x_i)$. Note that $P_{ji}(x_j, x_i) \ \equiv \ (P_{ji})^\smile(x_i, x_j)$, so we may replace both constraints by the single constraint $R(x_i, x_j)$ where R is a symbol for the relation $(P_{ij} \cap P_{ji}^\smile)$. It is trivial to show that $\langle a, b\rangle \in P_{ij}$ and $\langle b, a\rangle \in P_{ji}$ if and only if $\langle a, b\rangle \in R$. (if the reader will forgive the terminological abuse of identifying a relation with its symbol).



$P_{12}(x_1, x_2) \wedge P_{13}(x_1, x_3) \wedge P_{14}(x_1, x_4) \wedge P_{23}(x_2, x_3) \wedge P_{24}(x_2, x_4) \wedge P_{34}(x_3, x_4)$
(No unary constraints are shown)

Figure 1: A regular BCN in 4 variables

Thus the new BCSP is both regular, and has the same set of solutions as the original BCSP. Henceforth we shall assume that the BCSPs are all regular, and use the conventions that P_{ij} appears as a constraint if $i < j$, and that the constraint on the edge $[j, i]$ with $j > i$ is $(P_{ij})^{\smile}$.

2.2 Normal BCSPs

A BCSP is *separated* if no constraint on any pair of variables is the identity relation. To every BCSP corresponds a separated BCSP with the same set of solutions, using the following algorithm. Suppose P_{ij} is the identity relation, and $i < j$. Omit the constraint $P_{ij}(x_i, x_j)$ and replace every pair of constraints (where $k > j$) $P_{ik}(x_i, x_k)$ and $P_{jk}(x_j, x_k)$ by the single constraint $R(x_i, x_k)$, where R is a symbol for the relation $(P_{ik} \cap P_{jk})$ (and *mutatis mutandis* for the cases $k < i$ and $i < k < j$). It is straightforward to show how solutions to the new BCSP and solutions to the old BCSP are related by simply copying values for x_i as values also for x_j. A BCSP is *normal* if it is regular and separated. We use the same terminology for BCNs.

2.3 Path Consistency

Given binary relations $R(x, y)$, $S(x, y)$, the composition $R \circ S$ of the relations is defined by

$$(R \circ S)(x, y) \Leftrightarrow (\exists z)(R(x, z) \,\&\, S(z, y))$$

So $(a, b) \in (R \circ S)$ if and only if there is a value c such that $(a, c) \in R$ and $(c, b) \in S$. Now, for any satisfying values $a_1, ..., a_n$ for the variables $x_1, .., x_n$ we must have, by this definition, $\langle a_i, a_k \rangle \in P_{ik} \;\Rightarrow\; \langle a_i, a_k \rangle \in P_{ij} \circ P_{jk}$, for any $i, j, k \leq n$. This necessary

condition may be used as a pruning technique to narrow down the potential choices of a_i and a_k.

We give the definition for BCNs. A BCN A is said to be 3-*consistent* iff for any $i, j, k \leq n$ we have $P_{ik} \subseteq P_{ij} \circ P_{jk}$. It was shown in *[Mon74]* that 3-consistency is equivalent to the seemingly more general concept of path-consistency, so we shall follow standard practice and conflate the two notions. (Path-consistency says that the label on any edge in the network is contained in the composition along any path beginning at the tail of the edge and ending at the head. 3-consistency is just path-consistency for paths of length 2). The practical importance of pruning by path-consistency lies in its $O(n^3)$ complexity, and it has been widely used as a heuristic for satisfiability (which is NP-hard in many cases), as we shall note below *[Mon74, Mac77, Fre78, All83, MacFre85, MohHen86, VilKau86]*.

A *reduction* A' of a network A is a network with the same nodes such that the labels on each edge of A' are a subrelation of the labels on the corresponding edge of A. There is a *most-general path-consistent reduction*, $MGR_3(A)$, of any network A, i.e. a path-consistent reduction such that any path-consistent reduction of A is a reduction of it *[LadMad88.2]*. It follows that A and $MGR_3(A)$ have the same set of solutions. Most path-consistency algorithms compute $MGR_3(A)$, which may be accomplished in serial cubic time *[MacFre85, MohHen86]*. We shall show that $MGR_3(A)$ may be computed in parallel $O(n^2 \log n)$ time, and that there is a class of networks on which reduction-type algorithms for $MGR_3(A)$ (which include all published algorithms, and all parallel versions of them) take time $O(n^2)$.

2.4 Relation Algebras and BCNs

In *[LadMad88.2]*, we showed that the appropriate framework for developing the theory of BCNs is the *relation algebra* of Tarski. An *algebra of relations* is a collection of binary relations over a domain D (which may be defined simply as the union of the different domains of the relations) which is closed under the operations of sum (set-theoretic union), product (set-theoretic intersection), composition, converse (both defined above), and containing the empty relation (the empty set), the universal relation on D (the set $\{\langle x, y\rangle : x, y \in D\}$) and the identity relation (the set $\{\langle x, x\rangle : x \in D\}$). These operations are needed for roughly the following reasons. A general constraint in a BCSP is a *sum* of primitive relations; *product* is used in ensuring a BCSP is separated; *product* and *converse* are needed for ensuring a BCSP is regular; *product*, *converse* and *composition* are needed (as shown below) for computing $MGR_3(A)$; the *universal relation* is needed as a constraint on variables not otherwise constrained (note the definition of path-consistency requires that constraints on all possible pairs of variables be checked); if the *empty relation* constrains any pair of variables, then the BCSP is unsatisfiable (the possibility of this happening during the computation of $MGR_3(A)$ is the cubic time heuristic for satisfiability mentioned before); the *identity relation* is needed for identifying networks that are not separated. We note further that the subset relation is definable equationally using these operations, since $(P \subseteq R) \Leftrightarrow (P = (P \cap R))$.

Strictly speaking, we use symbols to represent the relations in a BCSP, and therefore we should use operations on the symbols which denote the corresponding operations on

the relations that the symbols denote. We shall use $+$ for $\cup$, $\cdot$ for $\cap$, $\smile$ for $\smile$, the semicolon ';' for $\circ$, and 1, 0 and 1' for the universal relation, empty relation and identity relation respectively (these symbols are traditional in relation algebra). We shall henceforth use these symbols in terms that label the edges of a BCN. Their denotations are the corresponding actual relations. We shall also use the symbol $\leq$ on terms for the partial order given by the subset relation on the relations.

An algebra of relations may be generated from any collection of binary relations by closing off under the operations in the usual manner. This algebra may be finite or infinite, depending on the relations one starts from. (Finiteness of the algebra should be distinguished from finiteness of the domain. There are many finite algebras with infinite domains, although of course the converse is impossible). If the algebra is finite, then there is always a collection of *atoms*, namely non-empty relations which are mutually disjoint, and such that any other relation in the algebra is a union of some of the atoms. (This follows from the fact that a relation algebra is a Boolean algebra with extra operators that distribute over sum, along with standard facts about Boolean algebras). These atoms form the primitive relations that many BCSPs are defined from. They are minimal non-zero elements in the ordering $\leq$ defined from the subset relation.

3 Algebraic Path-Consistency Algorithms

In this section, we formulate algorithms for path-consistency using the algebra developed so far. We give the general path-consistency algorithm in the form of a scheme, ignoring details of the iteration, because our purpose is to obtain an asymptotic result for all possible parallelisations. This scheme leads directly to the formulation as an algorithm over *relational matrices*, which give the upper bound result.

A test for path-consistency is a test whether $P_{ij} \leq (P_{ik} \; ; \; P_{kj})$ for every $i, j, k \leq n$. The following facts were all shown in *[LadMad88.2, LadMad88.3]*. The following algorithm scheme computes $MGR_3(A)$.

Path Consistency Algorithm: Given a BCN A, Iterate until no more changes: For every triangle (i, k, j) in A: do $P_{ij} \leftarrow P_{ij} \cdot (P_{ik} \; ; \; P_{kj})$.

The scheme allows iteration for a given triangle and enumeration over triangles to be combined in any way, serial or in parallel. This computation may be stopped if the label 0 ever appears on an edge, since the BCN is unsatisfiable, but we omit this minor modification here. Published algorithms all seem to have this structure, with attention paid to clever ways to enumerate the triangles. The computation of $MGR_3(A)$ may be regarded as computing the greatest fixed point of the following sets of equations (i.e. the collection of largest possible r_{ij} satisfying)

$$r_{ij} \leq P_{ij}$$

$$r_{ij} = r_{ij} \cdot \prod_k (r_{ik} \; ; \; r_{kj})$$

where the symbol $\prod_k$ denotes the product taken over all $k \leq n$. This observation leads to the representation of BCNs by $(n \times n)$ matrices of relation symbols, where the matrix

M corresponding to the BCN A has entries $M_{ij} = P_{ij}$. We call such a matrix a *relational matrix*. Relational matrix *product* and *composition* are defined by the following schemes

$$(M \cdot M')_{ij} = M_{ij} \cdot (M')_{ij}$$

$$(M \,;\, M')_{ij} = \prod_{k \leq n} M_{ik} \,;\, (M')_{kj}$$

Define also the power M^n by the usual induction,

$$M^1 = M,$$

$$M^{n+1} = (M^n) \,;\, M$$

So, for example, $M^2 = (M \,;\, M)$. Then the relational matrix corresponding to $MGR_3(A)$ may be computed by the following algorithm

Matrix Reduction Algorithm: Repeat $M \longleftarrow M \cdot M^2$ until $M \leq M^2$.

The matrix algorithm gives to the upper bound result. However, the formulation in terms of triangles is actually a more general scheme than the matrix formulation, since implicit in the matrix algorithm is that the intermediate matrices should be computed, whereas there is no such restriction in the triangle formulation. We refer to the triangle formulation as 'the scheme'.

Intermediate Normalisation We note in passing that it may be practically useful to apply the normality algorithm at specific points during a computation of $MGR_3(A)$. This issue is tangential to our immediate concerns.

3.1 The Upper Bound Result

We first discuss the complexity of the multiplication step. The formal similarity of relational matrix composition to a normal matrix product leads to the estimate of parallel $O(log\, n)$ time for a single matrix composition calculation, by the usual argument. This result for matrix multiplication would require up to n^3 processors. However, by a counting argument due to Steve Omuhundro *[Omo89]*, we may reduce the number of processors to $O(n)$, by 'folding' some of the computation, since the computation involves at most $O(n^3)$ discrete operations. This means that the algorithm is indeed practical, since it is reasonable to require as many processors as we have nodes in the BCN.

Now we consider the iteration step. The reduction algorithm iterates if a label is changed as a result of the composition. Labels may change only to smaller labels in the algebra, and since each label is a sum of atoms, it is easy to see that the number of such smaller labels is bounded by the number of atoms if the algebra is finite, and is therefore independent of the number of variables in the BCSP *[LadMad88.2]*. This bounds the number of iterations on which a given label may change. There are n^2 such labels, and therefore the algorithm may proceed through up to $O(n^2)$ iterations, leading to a parallel upper bound of $O(n^2 log\, n)$ for the computation of $MGR_3(A)$. We don't bother to analyse the iteration more thoroughly, since our class of hard examples in the next section will show that in some cases $\theta(n^2)$ steps are necessary.

4 A Class of Hard Examples

In this section, we give a class of hard examples for algorithms of the kind we have been considering. First, we attempt to loosely classify the algorithms by discussing the characteristics that make algoithms susceptible to the hard examples. We call these algorithms *reduction-type.*

Reduction-Type Algorithms Loosely speaking, we call an algorithm a reduction-type algorithm if it computes $MGR_3(A)$ by successive intermediate reductions. Our discussion here is somewhat vague, but important in that we need to note the characteristics of algorithms that will be susceptible to the hard examples. Someday, someone might invent a computation that is not equivalent to a reduction-type algorithm, but so far published algorithms for $MGR_3(A)$ seem to be equivalent to reduction-type algorithms. Since reduction-type algorithms must replace labels in the algebra by other labels in the algebra, to every reduction-type algorithm there corresponds an instance of the scheme, in the sense that there is some enumeration of triangles (with repetition) such that with that enumeration the scheme passes through the same intermediate steps as the reduction-type algorithm. Furthermore, the steps of a reduction-type algorithm that do not result in intermediate stages of the scheme may be omitted as superfluous, since one might as well go to the lowest label necessitated by a given triangle-consistency check. We shall therefore formally identify reduction-type algorithms with instances of the scheme for the purposes of this paper. Reduction-type algorithms may be serial or parallel, and for practical purposes parallelism can give great speedups in computations. For example, one may formally assign a processor to each edge $[i, j]$ of the BCN, which computes $P_{ij} \leftarrow P_{ij} \cdot \prod_k (P_{ik}; P_{kj})$. The speed of such a computation depends on the topology of the processor network, as well as decisions concerning eager or lazy evaluation.

4.1 The $\theta(n^2)$ Examples

In order to define the example networks, we first have to define the algebra of relations which provides the relation labels for the networks. So, first we define a certain algebra of relations $\mathcal{A}$. We shall not present the relations explicitly, but instead give the atoms, the converses of the atoms, and the compositions of atoms. This suffices to define to define any relation algebra *[Tar41, LadMad88.1, LadMad88.2]*, and it does not matter to us what the actual relations are, since our interest is in the hard examples they produce, not in the relations themselves. We then define a class of matrices N_k for k any positive integer which is a multiple of 5, all similar, with constraints from $\mathcal{A}$, such that any reduction-type algorithm passes through $O(k^2)$ intermediate networks while computing $MGR_3(N_k)$.

The Algebra $\mathcal{A}$. The atoms of $\mathcal{A}$ are $1'$, a, b, and c. Every atom is *symmetric*, i.e. $x^\smile = x$ for every atom x. It follows from general relation algebra that every element in the algebra must be symmetric. We define $0'$, the diversity element, to be the complement of $1'$, i.e. $(1')^-$ which is $a+b+c$ in $\mathcal{A}$. We define relation composition on the atoms thus:

$$a\,;\, b = b\,;\, a = b\,;\, c = c\,;\, b = 0' = a + b + c$$

$$b\,;\, b = c\,;\, c = 1$$

$$a\,;\,c = c\,;\,a = b + c$$

$$a\,;\,a = c^{-} = 1' + a + b$$

Since converse and composition have been defined over the atoms, they are thus defined over the whole algebra, since both of these operations distributes over sum *[LadMad88.2]*. We shall need to know the following compositions. Firstly, $(a + c)\,;\,(a + c) \geq (c\,;\,c) = 1$, since composition commutes with $+$. Similarly, from the table $(a+c)\,;\,a = a\,;\,(a+c) = 1$, $(a + b)\,;\,(a + c) = (a + c)\,;\,(a + b) = 1$, and $(a + b)\,;\,(a + b) = 1$.

The Examples N_k. N_k is defined as follows, for k a multiple of 5, say $k = 5m$. The nodes fall into 5 types of m nodes each, namely $v_1, ..., v_n$, $w_1, ..., w_n$, $x_1, ..., x_n$, $y_1, ..., y_n$, $z_1, ..., z_n$. The entries in N_k are

$$N(v_1, w_1) = a$$

$$N(v_i, v_{i+1}) = a, \quad i = 1, ..., m-1$$

$$N(w_i, x_i) = a, \quad i = 1, ..., m$$

$$N(x_i, y_i) = a, \quad i = 1, ..., m$$

$$N(y_i, z_i) = a, \quad i = 1, ..., m$$

$$N(z_i, w_{i+1}) = a, \quad i = 1, ..., m-1$$

$$N(w_i, v_j) = a + c, \quad i, j = 1, ..., m$$

$$N(y_i, v_j) = a + c, \quad i, j = 1, ..., m$$

$$N(x_i, v_m) = a + c, \quad i = 1, ..., m$$

$$N(z_i, v_1) = a + c, \quad i = 1, ..., m$$

For all other distinct nodes s, t, $N(s, t) = a + b$, and for any node s, $N(s, s) = 1'$. Figure 2 illustrates a way to think of an arrangement of the nodes. Not all the edges are drawn. The horizontal edges shown, and the diagonal edge $[v_1, w_1]$, are labelled with a. The z_i are each connected to v_1 with an edge labelled $(a + c)$, as are the x_i to v_4, and the w_i and y_i to each of $v_1, ..., v_4$. All other edges are labelled with $(a + b)$. We call a triangle *closed* (*[Mon74]*) if it is path-consistent as a subnetwork. There is precisely one nonclosed triangle in the network, namely (v_1, v_2, w_1). Closing this triangle changes the label on edge $[w_1, v_2]$ to a, making the neighboring triangle (v_2, v_3, w_1) now nonclosed, in an isomorphic manner. The computation proceeds through the triangles, closing one triangle and in so doing making a single neighbour nonclosed. Edges change from labels $(a + c)$ to label a, one at a time per iteration, so that at each intermediate state there is just one non-consistent triangle, with two edges labelled a, and one edge labelled $(a + c)$. The seed triangle is $\{v_1, v_2, w_1\}$, and the edges change in the following order.

$$[w_1, v_2], ..., [w_1, v_4], [x_1, v_4],$$

$$[y_1, v_4], ..., [y_1, v_1], [z_1, v_1],$$

$$[w_2, v_1], ..., [w_2, v_4], [x_2, v_4],$$

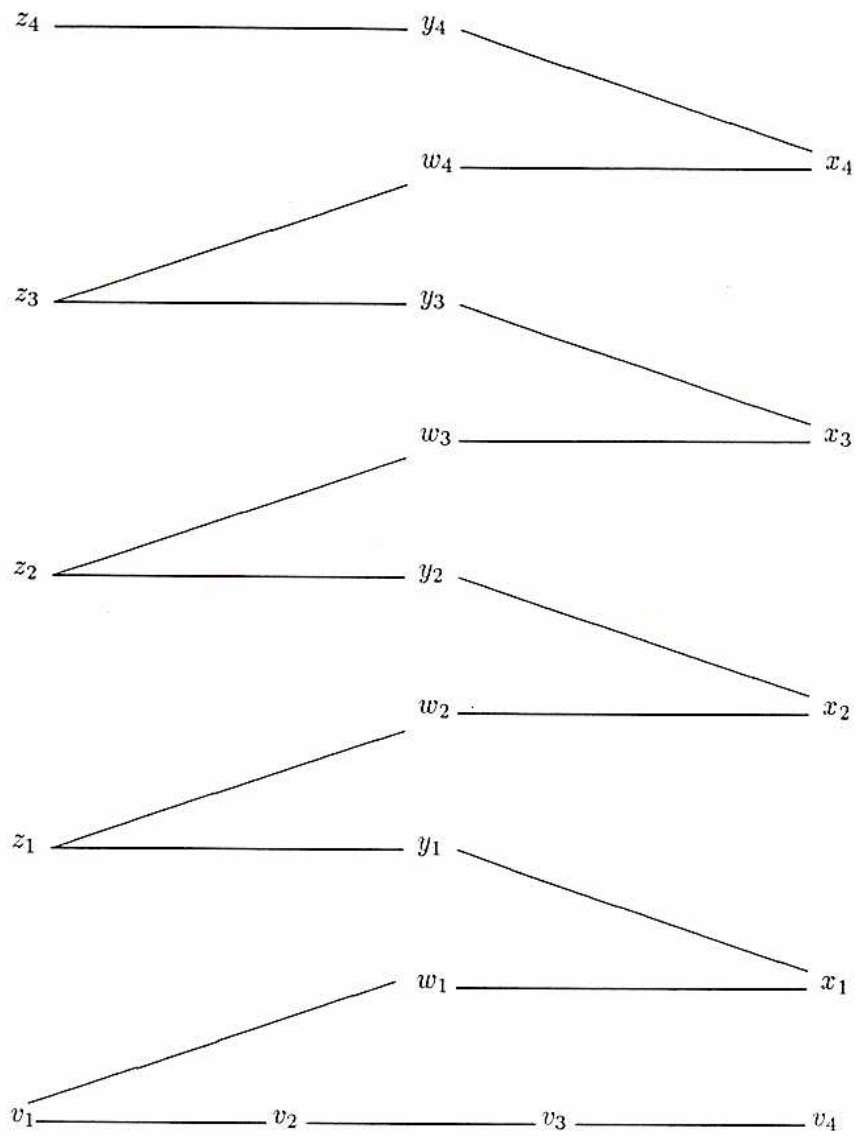
$$[y_2, v_4], ..., [y_2, v_1], [z_2, v_1],$$

.....

In the general case, replace 4 by m, and the number of edge-label changes is $(2m(m+1)-1)$. Thus there are $(2m(m+1)-1)$ intermediate states, so path consistency for this class of examples is $\theta(n^2)$ for reduction-type algorithms, parallel or serial.

5 Summary

We have introduced an algebraic approach to binary Boolean constraint satisfaction problems, and have used algebraic techniques to obtain an algorithm which operates in parallel $O(n^2\ log\ n)$ time on n processors, where n is the number of variables in the CSP. We have also given a class of hard examples for CSP algorithms, on which algorithms known so far, and all possible parallelisations of them (including ours) take time $\theta(n^2)$. We have also attempted to discuss characteristics of the reduction-type algorithms that make them susceptible to the hard examples.



All the lines shown are labelled with a.
All the z_i are connected to v_1 (vertically), labelled $(a + c)$, not shown.
All the x_i are connected to v_4 (vertically), labelled $(a + c)$, not shown.
All the w_i and y_i are connected to all the v_j, labelled $(a + c)$, not shown.
All edges not shown are labelled with $(a + b)$.

Figure 2: The arrangement of N_k for $m = 4$

Bibliography

All83 : Allen, J.F., *Maintaining Knowledge about Temporal Intervals*, Comm. A.C.M. 26 (11), November 1983, 832-843.

DecPea88 : Dechter, R., and Pearl, J., *Network-Based Heuristics for Constraint-Satisfaction Problems*, Artificial Intelligence 34, 1988, 1-38.

Fre78 : Freuder, E.C., *Synthesizing Constraint Expressions*, Communications of the ACM 21 (11), Nov 1978, 958-966.

LadMad88.1 : Ladkin, P.B., and Maddux, R.D., *Representation and Reasoning with Convex Time Intervals*, Kestrel Institute Technical Report KES.U.88.2, also submitted for publication.

LadMad88.2 : Ladkin, P.B., and Maddux, R.D., *On Binary Constraint Networks*, Kestrel Institute Technical Report KES.U.88.8.

Mac77 : Mackworth, A.K., *Consistency in Networks of Relations*, Artificial Intelligence 8, 1977, 99-118.

Mac87 : Mackworth, A.K., *Constraint Satisfaction*, in the *Encyclopedia of Artificial Intelligence*, ed. S. Shapiro, Wiley Interscience 1987.

MacFre85 : Mackworth, A.K., and Freuder, E.C., *The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems*, Artificial Intelligence 25, 65-74, 1985.

MohHen86 : Mohr, R., and Henderson, T.C., *Arc and Path Consistency Revisited*, Artificial Intelligence 28, 1986, 225-233.

Mon74 : Montanari, U., *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*, Inform. Sci. 7, 1974, 727-732.

Omo89 : Omohundro, S.M., *personal communication*, 1989.

Tar41 : Tarski, A., *On the Calculus of Relations*, Journal of Symbolic Logic 6, 1941, 73-89.

VilKau86 : Vilain, M., and Kautz, H., *Constraint Propagation Algorithms for Temporal Reasoning*, Proceedings of AAAI-86, 377-382, Morgan Kaufmann, 1986.