# Parallel and Distributed Algorithms for Constraint Networks

by
Ying Zhang and Alan K. Mackworth

Technical Report 91-6
May 1991

Department of Computer Science
The University of British Columbia
Vancouver, B. C. V6T 1W5
Canada

email: zhang@cs.ubc.ca, mack@cs.ubc.ca

# Parallel and Distributed Algorithms for Constraint Networks

Ying Zhang and Alan K. Mackworth*

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada, V6T 1W5
Email: zhang@cs.ubc.ca, mack@cs.ubc.ca

**Abstract**

This paper develops two new algorithms for solving a finite constraint satisfaction problem (FCSP) in parallel. In particular, we give a parallel algorithm for the EREW PRAM model and a distributed algorithm for networks of interconnected processors. Both of these algorithms are derived from arc consistency algorithms which are preprocessing algorithms in general, but can be used to solve an FCSP when it is represented by an acyclic constraint network. If an FCSP can be represented by an acyclic constraint network of size $n$ with width bounded by a constant then (1) the parallel algorithm takes $O(\log n)$ time using $O(n)$ processors and (2) there is a mapping of this problem to a distributed computing network of $poly(n)$ processors which stabilizes in $O(\log n)$ time.

## 1 Introduction

A Finite Constraint Satisfaction Problem (FCSP) can be informally described as follows. Given a set of variables, each with a finite domain, and a set of constraints, each specifying a relation on a subset of the variables, find the relation on the set of all the variables

*Shell Canada Fellow, Canadian Institute for Advanced Research

which satisfies all the given constraints simultaneously. FCSPs are useful abstractions of many problems in image understanding, planning, scheduling, database retrieval and truth maintenance [Mac87] [Dec91]. However, it is well known that the FCSP decision problem is $NP$-complete. In order to cope with the intractability of FCSPs, two strategies have been followed: (1) finding efficient algorithms for preprocessing, such as arc consistency [Mac77], path consistency [Mon74] and k-consistency [Fre78] algorithms and (2) exploiting the topological features of FCSPs to guide efficient algorithms for solving these problems [Dec91]. In this paper, we develop an approach to combining these two strategies. We generalize the binary arc consistency problem [Mac77] to an arc consistency problem on any constraint network. A class of arc consistency (AC) algorithms, which enforce arc consistency in a constraint network, will be studied in detail. AC algorithms are preprocessing algorithms in general, but can be used to solve an FCSP if it is represented by an acyclic constraint network. The complexity of AC algorithms for solving FCSPs is related to two parameters of acyclic constraint networks: *size* and *width*, which depend only on the topological features of FCSPs. The width of an acyclic constraint network, which is related to tree-width or armwidth [Jud90], induced width [DP89] and front length [Sei81], is one of the essential factors for the complexity. If an FCSP can be represented by an acyclic constraint network with width bounded by a constant, the problem can be solved in linear sequential time [MF85] [DP89] [RM89] [Fre90]. We will show that such a problem also has efficient parallel algorithms in the PRAM model and the distributed message passing model.

Research on parallel and distributed FCSPs started very recently. Theoretically, Kasif in [Kas90] [KD91] proved that the arc consistency problem for a binary constraint network is in $NC$ if the constraint network is acyclic, but $P$-complete in general. Some work on the connectionist approach to constraint satisfaction has also been reported [Coo89] [Gue91], but worst case time is not essentially improved by massive parallelism. In this paper, we generalize Kasif's result on the parallel complexity of the arc consistency problem in acyclic constraint networks, from width bounded by 2 to width bounded by any constant.

Even though the PRAM model is theoretically elegant for studying parallel complexity, many parallel machines are designed as reconfigurable interconnected processors with distributed memory. Collin and Dechter in [CD91] gave a distributed computation model and concluded that a distributed uniform daemon does not work even for a tree-structured constraint network. In this paper, we use a message passing distributed model and develop a method for mapping a constraint network to a distributed computing network. A distributed AC algorithm is discussed in this model. We show that for an acyclic constraint network of size $n$ with width bounded by a constant (1) if the constraint network can be mapped to a distributed network whose topology is a tree of bounded degree, then the distributed computing network can stabilize in $O(D)$ time, where $D$ is the diameter of the distributed network and (2) there is a mapping of the constraint network to a distributed computing network consisting of $poly(n)$ processors which stabilizes in $O(\log n)$ time.

## 2 Properties of Constraint Networks

Many problems can be formalized as constraint satisfaction problems, which can be represented by constraint networks. In this section, we use the *Course Scheduling* ($CS$) problem as an example to illustrate the major ideas. $CS$ is a simplified version of the general timetabling problems [EIS76] [SS80]. $CS(N, n, k)$ can be informally stated as follows. Given a set of courses, $\{c_1, c_2, \ldots, c_N\}$, each of which can be scheduled in one of $k$ timeslots, and a set of students, $\{s_1, s_2, \ldots, s_n\}$, each of whom takes some of the courses, the problem is to find a timetable such that no two courses taken by any student are scheduled in the same timeslot. We will come back to this example later when we discuss the properties of constraint networks.

Formally, a *constraint*, written $r(R)$, can be considered as a relation $r$ on a relation scheme $R$ [Mai83]. A *relation scheme* $R$ is a set of variables, $\{v_1, v_2, \ldots, v_k\}$. Associated with each variable $v_i$ is a domain $d_i$. Let $d = d_1 \cup d_2 \ldots \cup d_k$. A *relation* $r$ on a relation scheme $R$ is a set of mappings, $\{t_1, t_2, \ldots, t_p\}$, from $R$ to $d$, with the restriction that if $t \in r$ then $t(v_i) \in d_i$. We call $r(R)$ a *universal constraint* if $r$ includes all the possible mappings from $R$ to $d$ with that restriction. Projection, join and semijoin are operations defined on constraints. Let $r(R)$ be a constraint and $X \subseteq R$. The *projection* of $r$ onto $X$, written $\Pi_X(r)$, is a relation on the relation scheme $X$, $\Pi_X(r) = \{t(X) | t \in r\}$, where $t(X)$ is the mapping restricted to $X$. The *join operation* of two constraints $r(R)$ and $l(L)$, written $r \bowtie l$, is a relation on the relation scheme $R \cup L$, $r \bowtie l = \{t(R \cup L) | \exists t_r \in r, t_l \in l, t(R) = t_r(R), t(L) = t_l(L)\}$. The *semijoin operation* of $r(R)$ and $l(L)$, written $r \triangleleft l$, is a relation on the relation scheme $R$, $r \triangleleft l = \Pi_R(r \bowtie l)$. Projection, join and semijoin are the basic operations in our algorithms.

Any FCSP can be represented by a constraint network. Graphically, a constraint network is a labelled hypergraph, in which nodes represent variables and arcs represent constraints. Formally,

**Definition 2.1** *Constraint Network* $CN \equiv \langle V, dom, A, con \rangle$ *where*

- $V$ *is a set of variables,* $\{v_1, v_2, \ldots, v_N\}$.
- *Associated with each variable* $v_i$ *is a finite domain* $d_i = dom(v_i)$.
- $A$ *is a set of arcs,* $\{a_1, a_2, \ldots, a_n\}$.
- *Associated with each arc* $a_i$ *is a constraint* $r_i(R_i) = con(a_i)$.

Let $C$ be the set of constraints of a constraint network $CN$, $C = \{con(a_i) | a_i \in A\}$. The hypergraph of $CN$ is called the *scheme* of $CN$ [Dec91], $scheme(CN) = \{R | r(R) \in C\}$.

Clearly, $CS$ can be represented by a constraint network $CN$ with $V = \{c_1, c_2, \ldots, c_N\}$, $dom(c_i) = \{1, 2, \ldots, k\}$, $A = \{s_1, s_2, \ldots, s_n\}$, and $con(s_i) = r_i(R_i)$ where $R_i$ is the set of courses which $s_i$ takes and $r_i = \{t | \forall c_p, c_q \in R_i, c_p \neq c_q \rightarrow t(c_p) \neq t(c_q)\}$.

A *solution* $s$ of a constraint network $CN$ is a mapping from the set of all variables to their corresponding domains which satisfies all the given constraints. Formally, $s \in sol(CN)$ iff $\forall r(R) \in C$, $s(R) \in r$. A constraint network $CN$ is *minimal* iff $\forall r(R) \in C$, $\Pi_R(sol(CN)) = r$. Two constraint networks $CN$ and $CN'$ are *equivalent*, written $CN = CN'$, iff $V = V'$, $dom = dom'$, $sol(CN) = sol(CN')$.

A constraint network is a *binary* constraint network iff $\forall r(R) \in C, |R| \leq 2$. Arc consistency in a binary constraint network has been defined in [Mac77]. Enforcing arc consistency in a constraint network $CN$ results in an arc consistent constraint network $AC(CN)$ such that $CN = AC(CN)$. Various arc consistency (AC) algorithms have been proposed and analyzed [MF85]. In the rest of this section, we give a generalized definition for arc consistency on any constraint network, and discuss the properties of constraint networks and the relationship between equivalent constraint networks.

The *dual network* $DN$ of a constraint network $CN$ can be considered as an alternative representation of an FCSP. $DN$ is a labelled undirectional graph, in which the nodes are the arcs of $CN$ labelled by constraints. A dual network can be regarded as a binary constraint network with constraints of equality. Formally, for any two nodes $a_i, a_j$ in $DN$, with $con(a_i) = r_i(R_i)$ and $con(a_j) = r_j(R_j)$, if $I = R_i \cap R_j \neq \emptyset$, then $e = (a_i, a_j)$ is an edge in $DN$. The label of $e$, denoted $L(e)$, is $I$. For any $t_i \in r_i$ and $t_j \in r_j$, $t_i$ and $t_j$ are consistent on $e$ iff $t_i(L(e)) = t_j(L(e))$; $e$ is directional arc consistent from $a_i$ to $a_j$ iff $\Pi_{L(e)}(r_i) \subseteq \Pi_{L(e)}(r_j)$; $e$ is arc consistent iff $e$ is arc consistent in both directions. A dual network is arc consistent iff all the edges are arc consistent. We say a constraint network is *arc consistent* iff its dual network is arc consistent. Clearly, the definition reduces to the definition given in [Mac77] for binary constraint networks.

A *join network* $JN$ of a constraint network is a subnetwork of the dual network $DN$, with redundant edges removed. Formally, for any two nodes $a_i$, $a_j$ in $JN$, with $con(a_i) = r_i(R_i)$, $con(a_j) = r_j(R_j)$, and $I = R_i \cap R_j \neq \emptyset$, if there is a path between $a_i$ and $a_j$ in $JN$, consisting of $\langle e_1, e_2 \ldots, e_l \rangle$, such that $\forall 1 \leq k \leq l$, $I \subseteq L(e_k)$, then $e = (a_i, a_j)$ is not an edge in $JN$, otherwise $e$ is an edge in $JN$. A dual network can have many join networks with different redundant edges removed. Consider a $CS$ example with $N = 7$, $n = 6$, $k = 4$ and $R_1 = \{c_1, c_2, c_3\}, R_2 = \{c_1, c_4\}, R_3 = \{c_4, c_5\}, R_4 = \{c_5, c_6\}, R_5 = \{c_2, c_6\}, R_6 = \{c_1, c_2, c_7\}$. Figure 1 shows the graphs of the constraint network, the dual network and two of its join networks for this example.

A join network is arc consistent iff all the edges are arc consistent. Clearly all the join networks of a dual network are equivalent, in the sense that a dual network is arc consistent iff any of its join networks is arc consistent. In other words, a constraint network is arc consistent iff any of its join networks is arc consistent.

If a binary constraint network $CN$ is acyclic, a tree, $AC(CN)$ is a minimal network [MF85]. Generalizing, we say a constraint network $CN$ is acyclic iff its hypergraph is acyclic, a hypertree [Mai83] [SS88]. On the other hand, $CN$ is acyclic iff its join networks are trees [Mai83]. Applying an AC algorithm to any of its join networks results in a minimal constraint network. In other words, $AC(CN)$ is a minimal constraint network if $CN$ is acyclic.

Since a constraint network $CN$ may not be acyclic in general, as in the example shown in Figure 1, the solutions for $CN$ can be computed in three steps. First, construct an acyclic constraint network $ACN$ which is equivalent to $CN$. Second, enforce arc consistency in $ACN$. Third, construct the solutions for $AC(ACN)$. Each solution can be efficiently constructed for an acyclic minimal constraint network. The first step is called *tree clustering.* A tree-clustering scheme $TC$ for a constraint network $CN$ is a set of relation schemes such

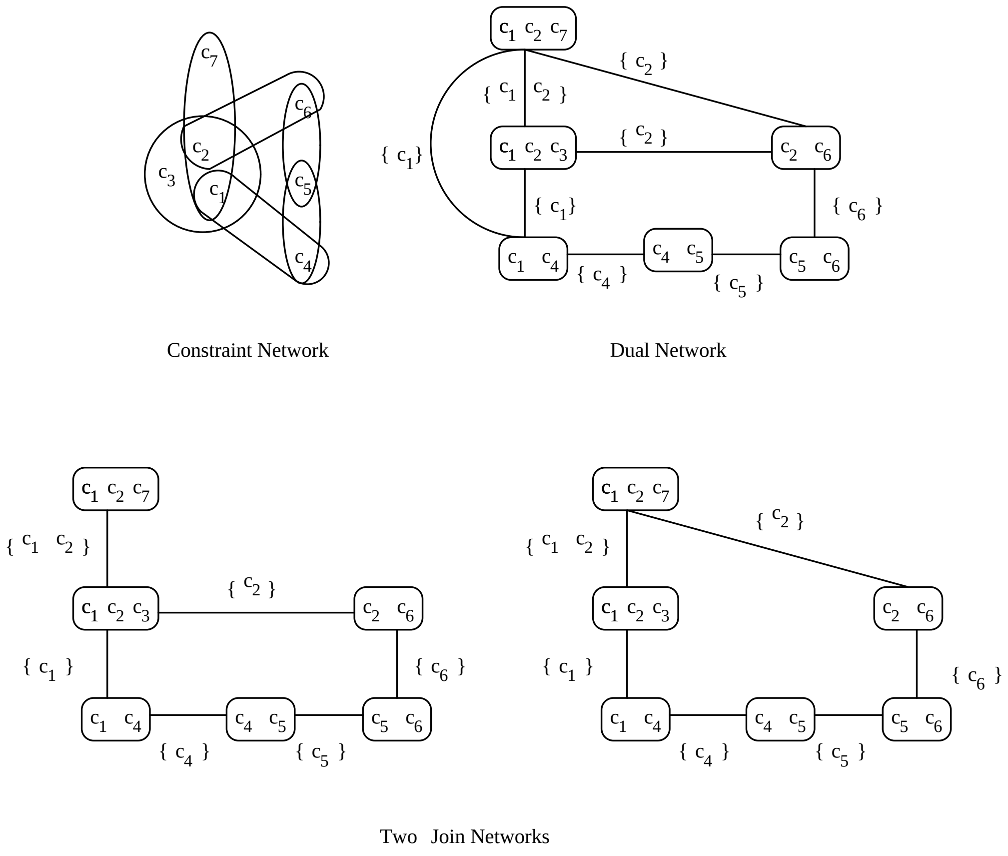


Figure 1: Constraint Network, Dual Network and Two Join Networks for the CS Example

that (1) $TC$ is a hypertree and (2) $\forall R \in scheme(CN), \exists R' \in TC, R \subseteq R'$. $TC$ can be obtained by applying a tree-clustering algorithm [DP89] to $scheme(CN)$. Figure 2 shows two different tree-clustering schemes for the constraint network given in Figure 1. Given a

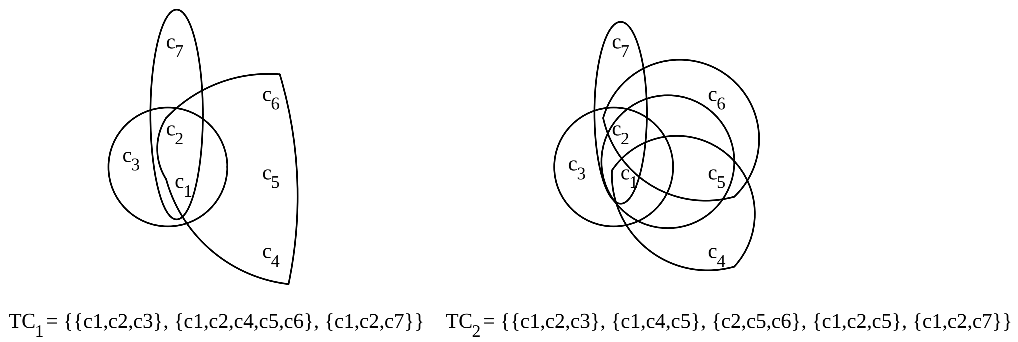


Figure 2: Tree Clustering Schemes

tree-clustering scheme $TC$ for $CN$, we can construct $ACN$ for $CN$ as follows. Let $ACN \equiv \langle V, dom, A \cup A', con' \rangle$ such that $\forall R' \in TC$ and $R' \notin scheme(CN), \exists a' \in A', con'(a') = r'(R')$ is a universal constraint and $\forall a \in A, con'(a) = con(a)$. It is easy to see that (1) $ACN = CN$ and (2) $ACN$ is an acyclic constraint network. For the $CS$ problem, if $TC$ is $TC_2$ in Figure 2 then $A' = \{a_1, a_2, a_3\}$, with universal constraints $con'(a_1) = r_1(\{c_1, c_4, c_5\})$, $con'(a_2) = r_2(\{c_2, c_5, c_6\})$, and $con'(a_3) = r_3(\{c_1, c_2, c_5\})$. A join network for this acyclic constraint network is shown in Figure 3.

The complexity of the arc consistency problem is related to two parameters of constraint networks, *size* and *width*. The size of a constraint network is the number of arcs, $size(CN) = |A|$. The width of a constraint network is the maximum size of the relation schemes, $width(CN) = \max_{R \in scheme(CN)}\{|R|\}$. For the constraint network $CN$ given in Figure 1, $size(CN) = 6, width(CN) = 3$. Its acyclic constraint network with tree-clustering scheme $TC_1$ has size 7, width 5; while its acyclic constraint network with tree-clustering scheme $TC_2$ has size 9 and width 3. For an acyclic constraint network of size $n$ and width

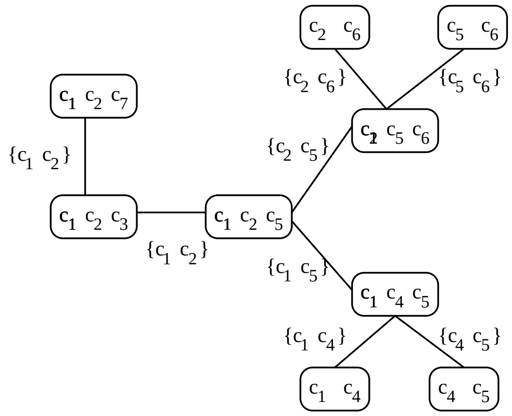


Figure 3: Join Network

$w$, arc consistency in any of its join networks takes $O(nl \log l)$ sequential time [DP89] where $l = m^w$ and $m = \max_{1 \leq i \leq N}\{|dom(v_i)|\}$. Since $w$ is the only exponential factor, it is critical for an acyclic constraint network to have small width. Even though finding a tree-clustering scheme with minimum width is an $NP$-complete problem [Fre90], there are many efficient algorithms for building sub-optimal tree-clustering schemes [DP89]. Furthermore, in many cases, the relation schemes are fixed, such as a relational database subjected to repeated queries, or have a regular topology such as an array, ring or mesh structure. The parallel and distributed AC algorithms assume that the equivalent acyclic network and its join network are constructed offline.

# 3 Parallel Algorithms and Complexity

Arc consistency on a binary acyclic constraint network is in $NC$ [KD91]. In this section, we generalize this result to any acyclic constraint network of bounded width. We show that, given a join network of an acyclic constraint network $CN$ of bounded width, there is an efficient parallel AC algorithm which takes $O(\log n)$ time using $O(n)$ processors in the

EREW PRAM model, where $n$ is the size of $CN$. One key idea is to apply the parallel tree contraction and expansion algorithm to the problem. The techniques of tree contraction and expansion are abstracted from many applications dealing with trees. Tree contraction reduces a tree to a single node, processing the information on the nodes as they are removed. Tree expansion is an inverse of contraction, propagating the information from the single node back to other nodes. It is known that there exist efficient parallel algorithms for tree contraction and expansion [MR85] [ADKP89]. We can obtain an efficient parallel algorithm for the problem by associating a procedure with each tree contraction and expansion step and proving that such a procedure executes in parallel quickly. The parallel algorithm is based on the parallel tree contraction algorithm in [MR85]. The procedures can be associated with other parallel tree contraction algorithms [ADKP89].

Let $T = \langle A, E \rangle$ be a rooted tree with nodes $A$ and edges $E$. A sequence of nodes $a_1, \ldots, a_k$ is called a *chain* if $a_{i+1}$ is the only child of $a_i$ for $1 \leq i < k$, and $a_k$ has exactly one child and that child is not a leaf. The parallel tree contraction algorithm defines two basic contract operations: RAKE and COMPRESS. RAKE is the operation of removing all leaves from $T$. COMPRESS is the operation on $T$ which contracts all the maximal chains of $T$ in half, by identifying $a_i$ with $a_{i+1}$ for $i$ odd, where $a_i$ is a node on a maximal chain. CONTRACT is the simultaneous application of RAKE and COMPRESS to the entire tree. After $\lceil \log_{5/4} n \rceil$ executions of CONTRACT on a tree of $n$ vertices, the tree is reduced to its root [MR85].

The parallel AC algorithm `ParAC` consists of two phases: `ContractAC` and `ExpandAC`. `ContractAC`, shown below, iterates tree contraction on a join network that is a rooted tree $T$. Semijoin operations are associated with each RAKE; join and projection operations are associated with each COMPRESS. The algorithm assumes that a tree-structured join network $T = \langle A, E \rangle$, with constraints associated with $A$ is allocated in common memory. For $a \in A$, let $pt(a)$ be the parent of $a$. If $a$ has only one child, let $cd(a)$ denote that child. If

$arg(a)$ is the number of children of $a$, let $chain(a)$ be a boolean function defined as $arg(a) = 1$ and $arg(pt(a)) = 1$. We call $p$ the *contracting parent* of $a$, if $a$ is raked from $p$ or $a$ is compressed to $p$. Let $cp(a)$ denote the contracting parent of $a$. Whenever a RAKE operation removes a leaf node with constraint $l(L)$ from its parent with constraint $r(R)$, a semijoin $r \triangleleft l$ is performed and $r$, the relation on the parent, is updated. Correspondingly for the COMPRESS operation, suppose $a_i, a_{i+1}$ are two consecutive nodes on a chain and let $a_{i-1}$ be the parent of $a_i$ and $a_{i+2}$ be the child of $a_{i+1}$ with $con(a_k) = r_k(R_k)$ and $L_k = R_k \cap R_{k+1}$, where $i-1 \leq k \leq i+1$. Whenever $a_i$ is identified with $a_{i+1}$, an operation $\Pi_{L_{i-1} \cup L_i \cup L_{i+1}}(r_i \bowtie r_{i+1})$ is applied.

```
Algorithm ContractAC: Tree Contraction Phase

Iterate the following procedure until T is
reduced to a single node, its root:

In Parallel for all a in A\{root}
begin
  r(R) := con(a); p(P) := con(pt(a));
  if {a has a leaf child} then /* RAKE */
  for {each leaf child c with constraint l(L)}
  begin
    r := r semijoin l; remove c; /* update links of a */
    cp(c) := a
  end
  else if (chain(a)) then /* COMPRESS */
  begin /* pt(a) is identified with a */
    create a new node a';
    c(C) := con(cd(a));
    p'(P') := con(pt(pt(a)));
    P" := C * R + R * P + P * P';/* + denotes union,* denotes intersection */
    p" := project (r join p) on P";
    con(a') := p"(P"); pt(cd(a)) := a'; cd(a') := cd(a);
    cd(pt(pt(a)) = a'; pt(a') = pt(pt(a));
    cp(a) := a'; cp(pt(a)) := a'
  end
end
```

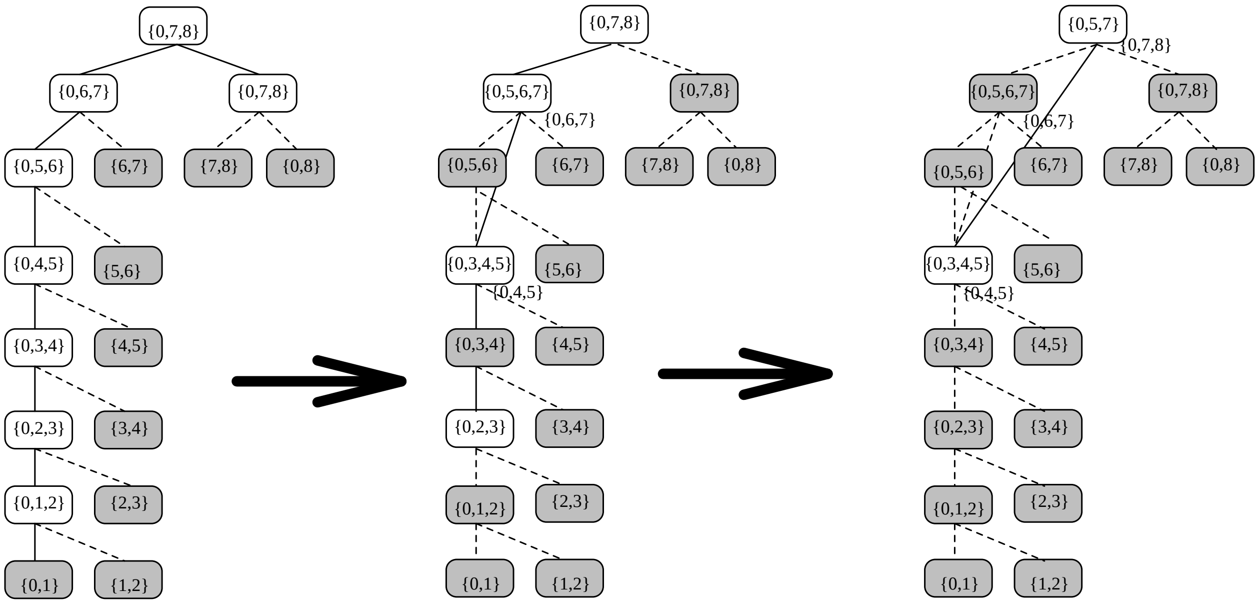


Figure 4: Tree Contraction Phase

Figure 4 shows the first three iterations of applying algorithm `ContractAC` to a join network, where shading depicts the removal of a node. It is clear that the number of iterations in `ContractAC` is identical to the number needed for CONTRACT.

During the tree contraction phase, links between a contracting parent and its contracted nodes are established. Let $T' = \langle A', E' \rangle$ be the join network resulting from applying `ContractAC` to $T$, such that $A' = A \cup A''$ where $A''$ includes all the nodes created in the tree contraction phase, and $(a, a') \in E'$ iff $a' = cp(a)$, i.e., $a'$ is the contracting parent of $a$. The tree expansion phase starts from the root node of $T'$ and propagates the solutions from root to leaves. Initially, the root is marked. Whenever the parent of a node is marked, the solutions can be computed for the node and then the node is marked.

The parallel AC algorithm `ParAC` simply applies `ContractAC` to $T$ and then applies `ExpandAC` to $T'$.

```
Algorithm ExpandAC: Tree Expansion Phase

marked(root) := 1;

Iterate the following procedure the same number
of times as for ContractAC:

In Parallel for all a in A' \{root} /* at most n nodes at each iteration */
begin
  if (marked(cp(a)) then
  begin
    r(R) := con(a); p(P) := con(cp(a));
    r := r semijoin p;
    marked(a) := 1
  end

Algorithm ParAC: Parallel Arc Consistency

input T, output T":
begin
   T' = ContractAC(T);
   T" = ExpandAC(T')
end
```

**Theorem 3.1** *The result of applying* `ParAC` *to $T$ is an arc consistent join network whose constraint network is minimal and equivalent to the constraint network of $T$.*

Proof: After the tree contraction phase, each edge $(a, a') \in E'$ is directional arc consistent from $a$ to $a'$. The tree expansion phase makes each edge in $T'$ arc consistent. Since $A \subseteq A'$, the constraint network with arcs $A$ is arc consistent. On the other hand, the constraints associated with $A'$ are derived from $A$. So the resulting constraint network of $T''$ is equivalent to the constraint network of $T$.□

**Theorem 3.2** *The algorithm* `ParAC` *takes $O(\log n)$ time using $O(n)$ processors in the EREW PRAM model, given a join network $T$ of an acyclic constraint network with bounded width.*

Proof: If the join network $T$ is not of bounded degree, it can be represented by a binary tree with at most twice as many nodes. Such a transformation takes $O(\log n)$ time in parallel [MR85]. So let $T$ be a binary tree join network. Let $w$ be the width of the acyclic constraint network represented by $T$. It is clear that RAKE does not change any of the relation schemes. After each COMPRESS operation, relation schemes are updated to $L_{i-1} \cup L_i \cup L_{i+1}$. But $|L_i|$ is always bounded by $w$, for all $i$, during the whole process. So the size of all relation schemes in $T'$ is bounded by $3w$. We also notice that since $T$ is a binary tree, RAKE can be done in constant time at each iteration. Thus the operations take constant time at each iteration of contraction. The total number of iterations is $\lceil \log_{5/4} n \rceil$. At each iteration, there are at most $n$ nodes which require at most $n$ processors. For the tree expansion phase, the tree sequence is the inverse of the sequence for tree contraction. There are no more than a bounded number of processors reading from the same memory location at any time. $\square$

The procedures associated with RAKE and COMPRESS for arc consistency can be associated with other parallel tree contraction algorithms. By associating semijoin with PRUNE and associating join and projection with BYPASS in the algorithm given by [ADKP89], arc consistency for an acyclic constraint network of bounded width can be done optimally in $O(\log n)$ time using $O(n/\log n)$ processors in an EREW PRAM.

# 4 Distributed Algorithms and Complexity

In the real world, many parallel machines are reconfigurable interconnected processors with distributed memory and asynchronous control. We define *MPD* as a *Message Passing Distributed* model:

**Definition 4.1 MPD model** *Each processor has a set of input and output ports. A processor can receive and send one message of bounded size, and perform one operation on operands of bounded size in its local memory at each step. The network consists of a set of* *processors connected by channels with any fixed topology. Communication is asynchronous with unbounded buffers and there is no cost for message passing.*

Let $s_i$ be a state of processor $i$. The *state* of a distributed computing network of $n$ processors is defined as $\langle s_1, s_2, \ldots, s_n \rangle$. A *stable* state $S$ of a network has the following property: if there is a time $t$ at which $S$ is the state then for all $t' > t$ $S$ is the state. A distributed algorithm on a network is stable if the network always achieves a stable state. The time complexity of a distributed algorithm is defined as the longest time required to achieve a stable state from any initial state.

The distributed constraint satisfaction algorithm `DistAC` is essentially the distributed version of `ParAC`. Let the nodes and edges of a join network $JN$ of a constraint network $CN$ map to processors and bidirectional channels in a distributed computing network, respectively. The algorithm is uniform: all processors have the same program. Let $r(R)$ be the local constraint and `propagate` be a subroutine for propagating the local constraint to its neighbors.

```
propagate:
    for {each channel c} send r(R) to c

Algorithm DistAC:
    propagate;
    loop
      begin
       s := r;
       for {each channel c}
       if {there is a message at channel c}
       begin
             receive r1(R1) from c;
             s := s semijoin r1;
       end
       if s =\= r then
          begin r := s; propagate end
      end
```

The following propositions characterize `DistAC`.

**Proposition 4.1** `DistAC` *is a stable distributed algorithm.*

Proof: This is obvious since semijoin is a monotone decreasing function on the size of the relations and the relations are initially finite.□

**Proposition 4.2** *If the width of constraint network $CN$ is bounded by a constant, the complexity of* `DistAC` *is $O(n)$, where $n = size(CN)$.*

Proof: In this case, the number of mappings in each relation is bounded by a constant $K$. So the total number of states is bounded by $Kn$. Therefore in $O(n)$ time the network will achieve a stable state. □

**Proposition 4.3** *A join network $JN$ is arc consistent iff the distributed network of $JN$ is stable.*

Proof: Obvious. □

It is clear that if a constraint network $CN$ is acyclic, $AC(CN)$ is minimal iff its corresponding distributed network is stable. On the other hand, such a distributed network tends to stabilize more quickly than arbitrary networks.

**Proposition 4.4** *If $JN$ is a join network of an acyclic constraint network of bounded width and $JN$ is of bounded degree, the complexity of* `DistAC` *is $\Theta(D)$ where $D$ is the diameter of $JN$.*

Proof: Let the degree of $JN$ be bounded by $K$. Consider $K$ time steps as one big time step. After $l$ big steps, any node may be affected by nodes at distance $l$. Since there is a unique path between any pair of nodes in a tree, a node can only be affected by some other node once. No node can be affected by any other node after $D$ big steps. So $KD$ is the upper bound. And it is obvious that $D$ is the lower bound, since two nodes at distance $D$ may affect each other. $\Box$

If the join network of an acyclic constraint network is of unbounded degree, we can transform the join network to a binary tree join network which can be mapped to a distributed network. Furthermore, it is easy to see that if the join network happens to be a balanced tree, then $D = O(\log n)$. However in many cases, a join network may be very unbalanced, with $D = \Omega(n)$. The following theorem shows that for any FCSP, if it can be represented by an acyclic constraint network $ACN$ of size $n$ and bounded width, then we can find a balanced binary tree join network, such that its acyclic constraint network, with size $poly(n)$ and bounded width, is equivalent to $ACN$.

**Theorem 4.1** *Let $n$ and $w$ be the size and width of an acyclic constraint network $ACN$. One can construct a balanced binary tree join network such that its acyclic constraint network $ACN'$ is equivalent to $ACN$ with $size(ACN') = poly(n)$ and $width(ACN') \leq 3w$.*

Proof: Let $JN$ be the join network of $ACN$ and $JN''$ be the binary tree representation of $JN$ and $ACN''$ be the acyclic constraint network of $JN''$. Let $n''$ and $w''$ be the size and width of $ACN''$. It is clear that $n'' \leq 2n$ and $w'' = w$. Let $L$ and $R$ be relation schemes. The following recursive algorithm $BT(T, L, R)$ takes a binary tree join network $T$ as input and returns the balanced binary tree join network.

If $T$ has only one node, return $T$. Otherwise do the following. First, find an edge in $T$ which is a "1/3 – 2/3" separator, i.e., it cuts the binary tree into two subtrees $T_1$ and $T_2$ with both sizes in the range of $[1/3n_T, 2/3n_T]$, where $n_T$ is the number of nodes in $T$. Let $BT(T_1, L, M)$ and $BT(T_2, M, R)$ be results of applying this algorithm recursively to $T_1$ and $T_2$ respectively, where $M$ is the label of the separator. Then create a node $C$ with a universal constraint on relation scheme $L \cup M \cup R$. Finally create a tree with $C$ as root, $BT(T_1, L, M)$ and $BT(T_2, M, R)$ as the left and right children of $C$, and return $C$.

Let $JN' = BT(JN'', \emptyset, \emptyset)$ be the result of applying the above algorithm to $JN''$. Let $ACN'$ be the acyclic constraint network of $JN'$. Since the height of $JN'$ is $\log_{3/2}(n'')$, there are at most $2^{\log_{3/2}(n'')}$ nodes, i.e., $size(ACN') = poly(n)$. Since all $|L|$, $|M|$ and $|R|$ are bounded by $w$, $width(ACN')$ is bounded by $3w$. $\square$

Thus, enforcing arc consistency in an acyclic constraint network of size $n$ with bounded width takes $O(\log n)$ time in a network of $poly(n)$ processors.

# 5 Conclusions

We have presented parallel and distributed algorithms for FCSPs. The analysis shown that for an FCSP that can be represented by an acyclic constraint network of bounded width, there are efficient algorithms in both parallel and distributed environments. The bounded width property of acyclic constraint networks characterizes a set of tractable FCSPs as well as efficiently parallelizable FCSPs. It is not generally true that a problem solvable in linear sequential time also has an efficient parallel algorithm, but it does happen to be the case for FCSPs.

**Acknowledgements**

We wish to thank Feng Gao, Nick Pippenger and Runping Qi for valuable suggestions and comments. The first author is supported by the University Graduate Fellowship from University of British Columbia. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

## References


[ADKP89] K. Abrahamson, N. Dadoun, D.G. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

[CD91] Z. Collin and R. Dechter. Distributed solution to the network consistency problem. In *AAAI-91 Spring Symposium on Constraint-Based Reasoning*, pages 174 – 181, 1991.

[Coo89] P. R. Cooper. Parallel object recognition from structure. Technical Report 301, Computer Science, University of Rochester, July 1989.

[Dec91] R. Dechter. Constraint networks: A survey. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*. Wiley, N.Y., 1991. (to appear).

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):257–388, April 1989.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Computer*, 5(4):691 – 703, 1976.

[Fre78] E. C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11), November 1978.

[Fre90] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceeding of AAAI-90*, 1990.

[Gue91] H. W. Guesgen. Connectionist networks for constraint satisfaction. In *Proc. 1991 Spring Symposium on Constraint-Based Reasoning*, pages 182 – 191, 1991.

[Jud90] J. S. Judd. *Neural Network Design and the Complexity of Learning*. MIT Press, 1990.

[Kas90] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.

[KD91] S. Kasif and A. L. Delcher. Analysis of local consistency in parallel constraint satisfaction networks. In *Proc. AAAI Symposium on Constraint Based Reasoning, Stanford*, pages 154 – 163, 1991.

[Mac77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mac87] A. K. Mackworth. Constraint satisfaction. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205 – 211. Wiley, N.Y., 1987.

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[MF85] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65 – 74, 1985.

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[MR85] G. L. Miller and J. H. Reif. Parallel tree contraction and its application. In *Proc. 26th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 478–489, 1985.

[RM89] F. Rossi and U. Montanari. Exact solution in linear time of networks of constraints using perfect relaxation. In *Proceedings First int. Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada*, pages 394–399, May 1989.

[Sei81] R. Seidel. A new method for solving constraint satisfaction problems. In *Proceeding of IJCAI-81*, pages 338–342, 1981.

[SS80] G. Schmidt and T. Strohlein. Timetable construction – an annotated bibliography. *Computer Journal*, 23(4):307 – 316, 1980.

[SS88] G. Shafer and P. P. Shenoy. Local computation in hypertrees. Technical report, School of Business, University of Kansas, August 1988. Working paper 201.