Parallel and Distributed Finite Constraint Satisfaction: Complexity, Algorithms and Experiments

by
Ying Zhang and Alan K. Mackworth

Technical Report 92-30
November 1992

Department of Computer Science
The University of British Columbia
Vancouver, B. C. V6T 1Z4
Canada

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

Parallel and Distributed Finite Constraint Satisfaction: Complexity, Algorithms and Experiments

Ying Zhang and Alan K. Mackworth*

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada, V6T 1Z4
Email: zhang@cs.ubc.ca, mack@cs.ubc.ca

Abstract

This paper explores the parallel complexity of finite constraint satisfaction problems (FCSPs) by developing three algorithms for deriving minimal constraint networks in parallel. The first is a parallel algorithm for the EREW PRAM model, the second is a distributed algorithm for fine-grain interconnected networks, and the third is a distributed algorithm for coarse-grain interconnected networks. Our major results are: given an FCSP represented by an acyclic constraint network (or a join tree) of size n with treewidth bounded by a constant, then (1) the parallel algorithm takes $O(\log n)$ time using $O(n)$ processors, (2) there is an equivalent network, of size poly(n) with treewidth also bounded by a constant, which can be solved by the fine-grain distributed algorithm in $O(\log n)$ time using poly(n) processors and (3) the distributed algorithm for coarse-grain interconnected networks has linear speedup and linear scaleup. In addition, we have simulated the fine-grain distributed algorithm based on the logical time assumption, experimented with the coarse-grain distributed algorithm on a network of transputers, and evaluated the results against the theory.

*Shell Canada Fellow, Canadian Institute for Advanced Research

1 Introduction

A Finite Constraint Satisfaction Problem (FCSP) can be described informally 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 which satisfies all the given constraints simultaneously. FCSPs are useful abstractions of many problems in image understanding, planning, scheduling, database retrieval and truth maintenance [Mac92] [Dec92]. 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 [Dec92].

Arc consistency (AC) plays an important role in constraint network preprocessing, not only because it has linear complexity in the number of constraints in binary constraint networks, but also because it produces the minimal network when it is applied to a tree [MF85]. Similar results were proved in relational database theory [BFM+81], i.e. for acyclic networks pairwise consistency is sufficient for global consistency. In fact, pairwise consistency is exactly arc consistency on a join network and the scheme of an acyclic network is a hypertree [SS88]. On the other hand, by exploiting the topological structures, a subset of tractable FCSPs was identified [Fre90] [Dec92]. The scheme of this kind of FCSP is a particular kind of graph called a partial k-tree, i.e. a graph which can be embedded in a hypertree with treewidth bounded by a constant k. It has been shown that many NP-complete problems in graph properties can be decided in linear time for the set of partial k-trees given a tree-decomposition [AL91]. Furthermore, it has been discovered that many classes of graphs are in this set [Bod86], including series-parallel graphs, outerplanar graphs [Bod86] and graphs generated by context-free grammars [RM89] [Cou90]. The complexity of FCSPs given a tree-decomposition is related to two parameters of acyclic constraint networks: *size* and *treewidth*, which depend only on the topological features of FCSPs. The treewidth of a constraint network, which is equivalent to the minimum induced width [Dec92], is one of the essential factors for the complexity. If a tree-decomposition is given for an FCSP with treewidth bounded by a constant, the decision problem can be solved in linear sequential time [MF85] [Dec92] [RM89] [Fre90].

Research on parallel and distributed algorithms for FCSP started very recently. The parallel complexity of problems can be characterized by a class called NC [KR90]. A problem is in NC iff there is a parallel algorithm for this problem in a PRAM model, which takes polylog time using polynomial number of processors, i.e. theoretically, the problem can be solved efficiently in parallel. It has been shown [Bod88] that many graph recognition problems which are NP-complete in general are in NC when restricted to graphs with bounded treewidth. It is proved that the arc consistency problem for a constraint network of binary relations is in NC if the constraint network is a tree, but P-complete in general [Kas90]. It is well known that $NC \subseteq P$ and a P-complete problem is in NC iff $NC = P$ which is unlikely to be true. Some work on the connectionist approach to constraint satisfaction has also been reported [Coo89] [Gue91], but worst case time has not essentially been improved by massive parallelism.

Even though PRAM models are theoretically elegant for studying parallel complexity, many parallel machines are designed as reconfigurable interconnected processors with distributed memory. Such a distributed computation model for FCSPs has been given in [CD91]. The distributed problem solving with distributed constraint satisfaction has been formalized by [YDIK92]. A series of distributed arc consistency algorithms has been developed and the performance both in simulation and on real parallel machines has been analyzed. However, the complexity of distributed constraint satisfaction remained unknown.

The major contributions of this paper are three new parallel and distributed algorithms for constraint networks, and the analysis of these algorithms. Given an FCSP represented by an acyclic constraint network (or a join tree) of size n with treewidth bounded by a constant, we have the following results.

1. The parallel algorithm `PTAC` takes $O(\log n)$ time using $O(n)$ processors, therefore such an FCSP decision problem is in NC. It generalizes Kasif's result [Kas90] from treewidth 1 to any constant k; the result can be further generalized to FCSPs with treewidth bounded by $O(\log n)$.

2. There is an equivalent network, of size poly(n) with treewidth also bounded by a constant, which can be solved by the fine-grain distributed algorithm, `DJAC`, in $O(\log n)$ time using poly(n) processors.

3. The algorithm for coarse-grain interconnected networks, `DTAC`, has linear speedup and scaleup.

In addition to the theoretical results, we have simulated the fine-grain distributed algorithm based on logical time assumptions and experimented with the coarse-grain distributed algorithm on a network of transputers.

Table 1 summarizes current knowledge on the complexity of FCSPs; our results are marked with (†). The rest of the paper is organized as follows. Section 2 gives the definition of constraint networks and the related concepts, followed by two sequential `AC` algorithms, `JAC` and `TAC`. Section 3 presents the parallel algorithm `PTAC` for the EREW PRAM model, which yields the parallel complexity result. Section 4 develops the distributed algorithm `DJAC` for fine-grain interconnected networks, and related complexity results are developed for this

Table 1: The complexity of FCSPs

Problem	*Restriction*	*Sequential Complexity*	*Parallel Complexity*
FCSP decision	binary relations	NP-complete	
Enforcing Consistency	arc consistency	linear	P-complete
	k-consistency	poly	P-complete
FCSP decision given tree-decomposition	treewidth = 1	linear	NC
	treewidth $\leq k$	linear	NC †
	treewidth $\leq k \log n$	poly	NC †

model. Preliminary versions of the results in Sections 3 and 4 have been presented earlier [ZM91a, ZM91b, ZM91c]. Section 5 constructs the distributed algorithm `DTAC` for coarse-grain interconnected networks; speedup and scaleup are discussed for this model. Section 6 shows the experimental results both in simulation and on real distributed machines. Section 7 concludes the paper.

2 Properties of Constraint Networks

Many problems can be formalized as finite 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 [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.

2.1 *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 $R \cup L$, $r \bowtie l = \{t(R \cup L) | t(R) \in r, t(L) \in l\}$. 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 labeled hypergraph, in which nodes represent variables and arcs represent constraints. Formally, a *constraint network* is defined as follows.

Definition 2.1 (Constraint network) *A constraint network is a quadruple* $CN = \langle V, dom, A, con \rangle$ *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 [Dec92], $scheme(CN) = \langle V, E\rangle$ where $E = \{R|r(R) \in C\}$.

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, let $sol(CN)$ be the set of all solutions of CN, $s \in sol(CN)$ iff $\forall r(R) \in C$, $s(R) \in r$. A constraint network CN is *minimal* iff $\forall r(R) \in C$, $\Pi_R(sol(CN)) = r$. Clearly, the FCSP decision problem can be reduced to the problem of deriving minimal networks. Two constraint networks CN and CN' are *equivalent*, written $CN \sim CN'$, iff $V = V', dom = dom', sol(CN) = sol(CN')$. A constraint network is *binary* iff $\forall r(R) \in C, |R| \leq 2$.

2.2 *Dual networks and join networks*

The *dual network* DN of a constraint network CN is an alternative representation of an FCSP. DN is a labeled undirected graph, in which the nodes are the arcs of CN labeled by constraints. 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. A dual network can be regarded as a binary constraint network with constraints of equality.

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 scheme of the constraint network, the dual network and four of its join networks for this example. Two join networks JN_1 and JN_2 are *equivalent*, written $JN_1 \sim JN_2$ iff their correspondent constraint networks are equivalent.

2.3 *Acyclic constraint networks*

A constraint network CN is *acyclic* iff its scheme is acyclic, a hypertree [Mai83] [SS88]. It has been shown that a hypergraph is a hypertree iff its join graphs are trees [Mai83]. A constraint network may not be acyclic in general, as in the example shown in Fig. 1. However, for any hypergraph, a tree-decomposition can be defined as follows.

Definition 2.2 (Tree-decomposition) *Let $G = \langle V, E\rangle$ be a hypergraph. A tree-decomposition of G is a pair $\langle \{X_i | i \in I\}, T = \langle I, F\rangle\rangle$, with $X_i \subseteq V$ and T a tree, with the following properties:*

- $\bigcup_{i \in I} X_i = V$,
- *for every edge $e \in E$, there is an $X_i, i \in I$ such that $e \subseteq X_i$,*
- *for all $i, j, k \in I$, if j lies on a path in T from i to k, then $X_i \cap X_k \subseteq X_j$.*

The *treewidth* of a tree-decomposition $\langle \{X_i | i \in I\}, T\rangle$ is $\max_{i \in I} |X_i| - 1$. The treewidth of G, denoted $treewidth(G)$, is the minimum treewidth of a tree-decomposition of G, taken over all possible tree-decompositions of G. Given a constraint network $CN = \langle V, dom, A, con\rangle$, if $\langle \{X_i\}_{i \in I}, T\rangle$ is a tree-decomposition of $scheme(CN)$, $TC = \langle V, \{X_i\}_{i \in I}\rangle$ is called a *tree-clustering* scheme of CN. It is easy to see that (1) any tree-clustering scheme TC of CN is

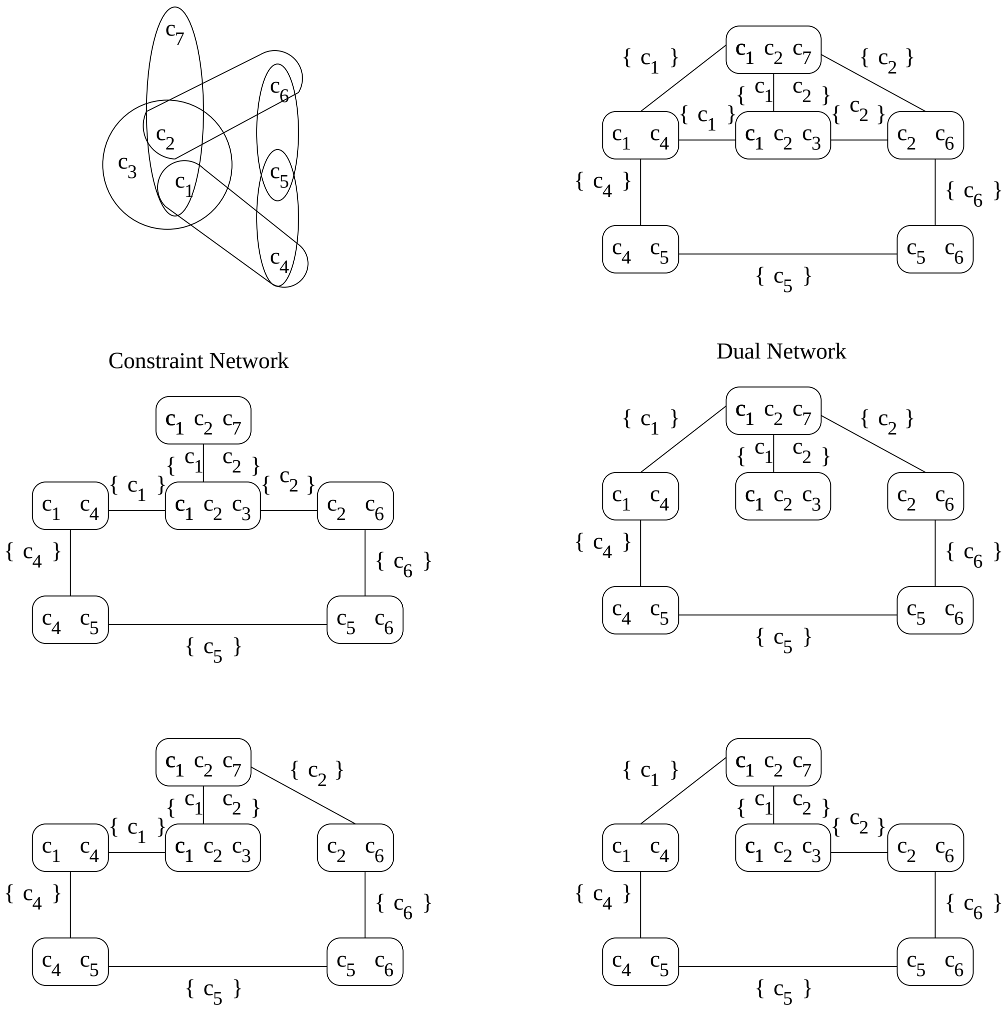

Figure 1: A constraint network, its dual network and four join networks

a hypertree and (2) for any relation scheme R in CN, there is an edge R' in TC, such that $R \subseteq R'$. TC can be obtained by applying a tree-clustering algorithm [Dec92] to $scheme(CN)$. Figure 2 shows two different tree-clustering schemes for the constraint network given in Fig. 1.

TC₁ = {{c1,c2,c3}, {c1,c2,c4,c5,c6}, {c1,c2,c7}}   TC₂ = {{c1,c2,c3}, {c1,c4,c5}, {c2,c5,c6}, {c1,c2,c5}, {c1,c2,c7}}

Figure 2: Two tree clustering schemes

Given a tree-clustering scheme TC for CN, we can construct an equivalent acyclic network ACN for CN as follows. Let $ACN = \langle V, dom, A \cup A', con' \rangle$ 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) $ACN \sim CN$ and (2) ACN is an acyclic constraint network. For the CS problem, if TC is TC_2 in Fig. 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 Fig. 3. We call a join network of an acyclic constraint network a *join tree*.

2.4 *Enforcing consistency*

A constraint network $CN = \langle V, dom, A, con \rangle$ is *pairwise consistent* [BFM+81] if for all pairs of $r_i(R_i)$ and $r_j(R_j)$ in C, $\Pi_{R_i \cap R_j}(r_i) = \Pi_{R_i \cap R_j}(r_j)$ where $C = \{con(a) | a \in A\}$. It is easy to see that a constraint network is pairwise consistent iff its dual network is arc consistent [Mac77].

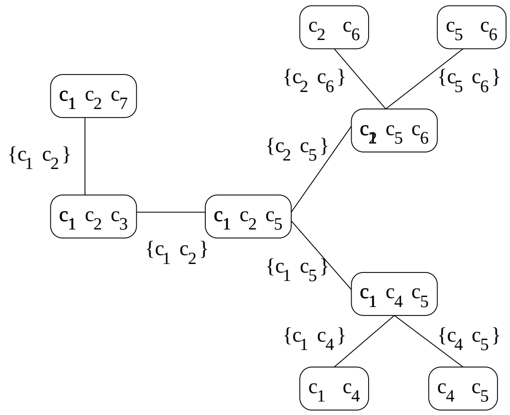

Figure 3: A join tree

Furthermore, the dual network is arc consistent iff its join networks are arc consistent. It is clear that if a constraint network is minimal, it is pairwise consistent. However, the converse is not always true for arbitrary networks. For any relations on the relation schemes in a constraint network, pairwise consistency implies global consistency if the constraint network is acyclic [BFM+81]. This is equivalent to the result in [MF85] that arc consistency enforces a minimal network if the network is a tree.

Let `AC` be a series of algorithms with a join network as input and an equivalent arc consistent join network as output. `AC` enforces arc consistency on a join network, i.e. `AC`$(JN) \sim JN$. We present two sequential `AC` algorithms: `JAC` and `TAC`.

`JAC` (Fig. 4) is a modified version of `AC-3` in [Mac77]. First put the set of arcs, pairs of constraints, in a queue. While the queue is not empty, do the following: Get an arc from the queue, perform a semijoin of the two constraints, with the first constraint against the second. If the first constraint is refined, make sure all of its neighboring arcs are on the queue. The algorithm finishes with an arc consistent join network. According to [MF85], `JAC` is linear in the number of arcs, or edges, in the join network. Furthermore, it can be more efficient for a join tree if the arcs in the queue are initially ordered.

```
Algorithm JAC: Enforce Arc Consistency
Input: join network <A,E>;
Output: arc consistent network;
BEGIN
   q := the set of all arcs in E;
/* if {a1,a2} in E, then both (a1,a2) and (a2,a1) in q */
   WHILE (q is not empty) DO
   BEGIN
     remove arc (a1,a2) from q;
     /* con(a1) = r1(R1),con(a2) = r2(R2) */
     r := r1 semijoin r2;
     IF r =\= r1 THEN
        BEGIN
           r1 := r;
           q := q union {(a,a1)|{a,a1} in E}\{(a2,a1)}
        END
   END
END
```

Figure 4: Enforcing arc consistency in a join network

TAC (Fig. 5) is an explicit version of JAC for rooted join trees (by picking any node as root), such that each arc is checked only once. The algorithm produces an arc consistent

```
Algorithm TAC: Enforcing Arc Consistency
Input: rooted join tree <A,E>;
Output: arc consistent join tree;
BEGIN
  q0 := the set of all nodes in A;
  /* ordered from children to parents */
  q := q0;
  WHILE (q is not empty) DO
  BEGIN
    remove node a from q; /* con(a) = r(R) */
    FOR (all (a,a1) in E) /* con(a1) = r1(R1) */
    /* a is the parent of a1 */
    r := r semijoin r1
  END
  q: = reverse q0;
  /* ordered from parents to children */
  WHILE (q is not empty) DO
  BEGIN
    remove node a from q;
    FOR (all (a1,a) in E) DO
    r := r semijoin r1
  END
END
```

Figure 5: Enforcing arc consistency in a join tree

join tree by traversing the join tree twice: first bottom up, then top down. The algorithm consists of two phases. The first phase starts from the leaves and each constraint performs semijoin operations with all of its children. This phase results a *directional* arc consistent network, i.e. parents are consistent with their children. The second phase starts from the root and each constraint performs a semijoin operation with its parent.

The solutions for a constraint network CN can be computed in three steps. First, construct a join tree JT of an acyclic constraint network ACN which is equivalent to CN. Second, enforce arc consistency in JT. Third, apply backtrack free search [Fre82] to $\mathtt{AC}(JT)$. In this paper, we concentrate on the parallel and distributed algorithms for the second step, while the first step is considered as preprocessing, in that our algorithms assume that CN is represented by JT (for `PTAC` and `DTAC`) or JN (for `DJAC`).

2.5 *Sequential complexity*

The complexity of the arc consistency problem on a join network is related to two parameters of its constraint network: *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 minus one, $width(CN) = \max_{R \in scheme(CN)} |R| - 1$. The width of an acyclic constraint network is also called the *treewidth*. For the constraint network CN given in Fig. 1, $size(CN) = 6$, $width(CN) = 2$. Its acyclic constraint network with tree-clustering scheme TC_1 has size 7, treewidth 4; while its acyclic constraint network with tree-clustering scheme TC_2 has size 9 and treewidth 2. The treewidth of a constraint network is the minimum treewidth over all of its acyclic constraint networks resulting from tree-decomposition.

For an acyclic constraint network of size n and treewidth w, arc consistency in any of its join trees takes $O(nl \log l)$ sequential time [Dec92] where $l = m^{w+1}$ 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 treewidth. Even though finding the treewidth of a constraint network and its corresponding tree-decomposition is an NP-complete problem [AL91], there are many efficient algorithms for building sub-optimal tree-clustering schemes [Dec92]. 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 off-line.

3 A Parallel Algorithm and Complexity

Arc consistency on a join tree whose constraint network has treewidth 1 is in NC [Kas90]. In this section, we generalize this result to any acyclic constraint network of bounded treewidth. We show that, given a join tree of an acyclic constraint network CN of bounded treewidth, 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.

The key idea is to apply parallel tree contraction and expansion algorithms 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].

3.1 *Parallel tree contraction*

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 $1 \le i < k$, 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 (Fig. 6). RAKE is the operation of removing all

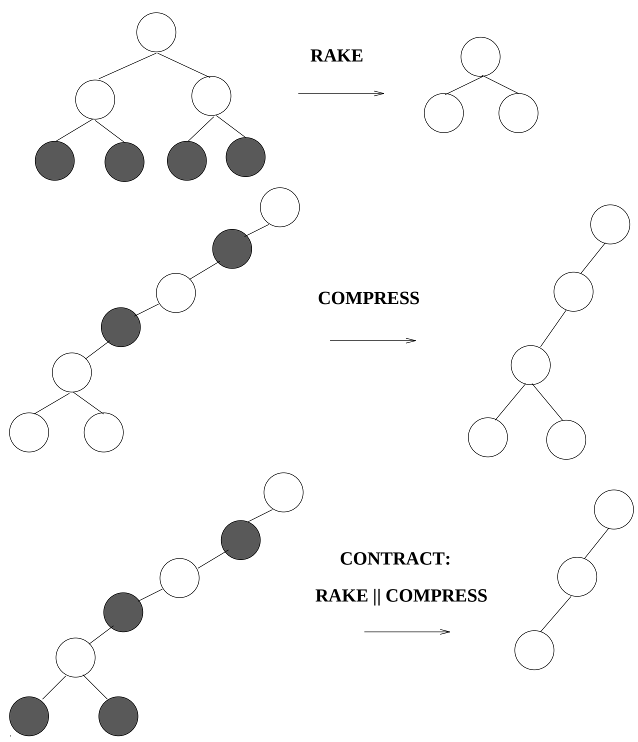

Figure 6: Parallel tree contraction

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].

3.2 *The parallel algorithm* `PTAC`

The parallel `AC` algorithm `PTAC` consists of two phases: `ContractAC` and `ExpandAC`. `ContractAC`, shown in Fig. 7, iterates tree contraction on a rooted join tree T. Semijoin operations are associated with each RAKE; join and projection operations are associated with each COMPRESS. The algorithm assumes that the join tree $T = \langle A, E \rangle$, with constraints associated with A, is allocated in the 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, to produce the constraint for the new merged node.

Figure 8 shows the first three iterations of applying algorithm `ContractAC` to a join tree resulting from a tree-decomposition of a constraint network with a ring topology, where shading depicts the removal of a node from T. 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 tree 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 (Fig. 9) 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 `PTAC` (Fig. 10) simply applies `ContractAC` to T and then

```
Algorithm ContractAC: Tree Contraction Phase
Input: rooted join tree T = <A,E>;
Output: directional arc consistent join tree;

Iterate the following procedure until T=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 7: The algorithm for parallel directional arc consistency: I

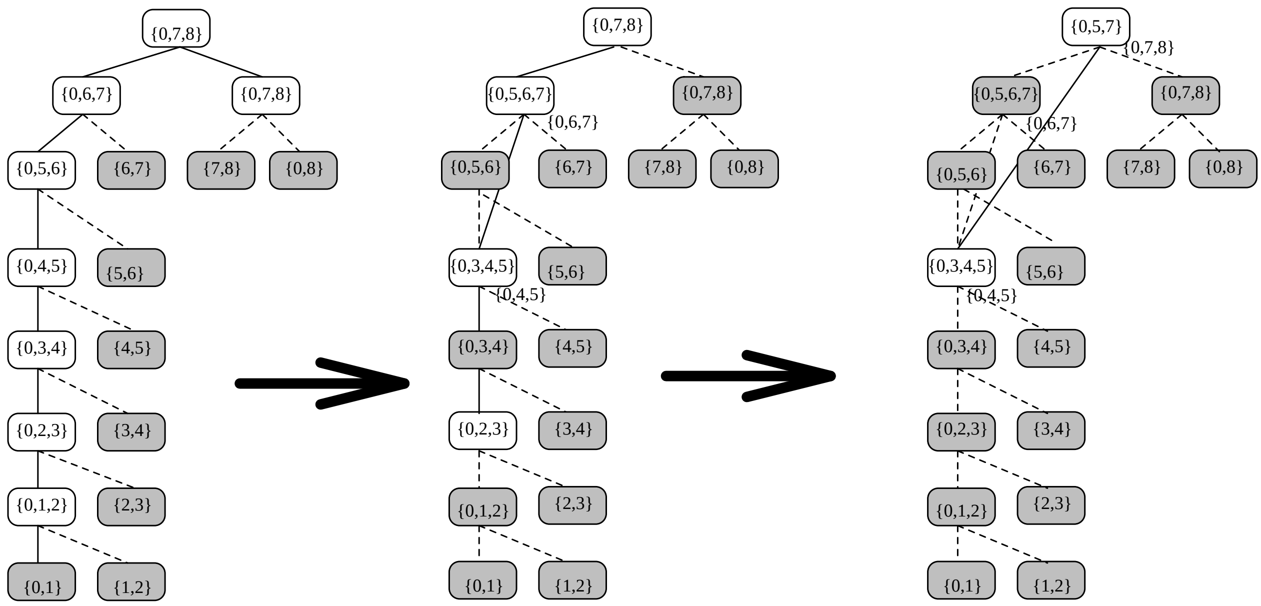

Figure 8: The tree contraction phase

applies `ExpandAC` to T'.

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

Proof: The result of the tree contraction phase is the same as the result of the first phase in `TAC`, i.e. the network is directional arc consistent. The result of the tree expansion phase is the same as the second phase in `TAC`, which 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.$\Box$

```
Algorithm ExpandAC: Tree Expansion Phase
Input: result of ContractAC T' = <A',E'>;
Output: arc consistent join tree;

marked(root) := 1;

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

In Parallel for 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
END
```

Figure 9: The algorithm for directional arc consistency: II

```
Algorithm PTAC: Parallel Arc Consistency
Input: rooted join tree T;
Output: arc consistent join tree T";
BEGIN
        T' = ContractAC(T);
        T" = ExpandAC(T')
END
```

Figure 10: The parallel `AC` algorithm `PTAC`

3.3 *Parallel complexity*

The parallel complexity of arc consistency on a join tree can be derived from the following theorem.

Theorem 3.1 *The algorithm* `PTAC` *takes* $O(\log n)$ *time using* $O(n)$ *processors in the EREW PRAM model, given a join tree of an acyclic constraint network with bounded treewidth.*

Proof: If the join tree 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, ADKP89]. So let T be a binary join tree. Let w be the treewidth of the acyclic constraint network of the join tree 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. Furthermore, there are no more than a bounded number of processors reading from the same memory location at any time. $\Box$

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 treewidth can be done optimally in $O(\log n)$ time using $O(n/\log n)$ processors in an EREW PRAM.

Since an acyclic constraint network whose join tree is arc consistent is also a minimal network, finding a minimal network (the FCSP decision problem) for a bounded treewidth constraint network is in NC given the tree-decomposition.

The result can be extended to constraint networks with the treewidth bounded by $O(\log n)$. It is easy to see that the number of tuples in a relation scheme of size $O(\log n)$ is bounded by $poly(n)$. Join and projection operations can be considered as variations of sort and merge operations which can be done $O(\log l)$ using $O(l)$ processors, where l is the length of the longer list [KR90]. Therefore, each RAKE and COMPRESS operation in `PTAC` takes $O(\log n)$ using $poly(n)$ processors. Since there $O(\log n)$ iterations, the algorithm takes $O(\log^2 n)$ using $poly(n)$ processors.

4 A Distributed Algorithm and Complexity

In the real world, many parallel machines are reconfigurable interconnected processors with distributed memory and asynchronous control. We define a fine-grain interconnected network (FIN) model as follows:

Definition 4.1 (FIN 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 message passing is of bounded delay.*

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.

4.1 *The distributed algorithm* DJAC

The distributed constraint satisfaction algorithm `DJAC` (Fig. 11) is essentially the distributed version of `JAC`. Let the nodes and edges of a join network 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. The following propositions characterize the properties of `DJAC`.

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

Proof: This is obvious since semijoin is a monotone decreasing function on the number of the relation tuples which is initially finite.□

Proposition 4.2 *A join network JN is arc consistent iff the distributed network of JN is stable.*

Proof: Obvious. □

4.2 *The distributed complexity*

The distributed complexity of arc consistency can be derived from the following propositions.

Proposition 4.3 *If the width of constraint network CN is bounded by a constant, the complexity of* `DJAC` *is $O(n)$, where $n = size(JN)$.*

Proof: Suppose the number of relation tuples on each relation scheme is bounded by a constant K. So the total number of messages is bounded by $2Kn$. Therefore in $O(n)$ time the network will achieve a stable state. □

```
propagate:
    FOR (all channel c) send r(R) to c

Algorithm DJAC: Distributed AC
Input: join network <A,E>;
Output: arc consistent network;
BEGIN
   propagate;
   LOOP
   BEGIN
       s := r;
       FOR (all channel c)
       IF (there is a message at channel c) THEN
       BEGIN
           receive r1(R1) from c;
           s := s semijoin r1
       END
       IF s =\= r THEN
           BEGIN r := s; propagate END
   END
END
```

Figure 11: The distributed AC algorithm DJAC

It is clear that given a join tree of an acyclic constraint network, the resulting constraint network is minimal iff its corresponding distributed network is stable. Moreover, such a distributed network tends to stabilize more quickly than an arbitrary network.

Proposition 4.4 *If JT is a join tree of an acyclic constraint network of bounded treewidth and JT is of bounded degree, the complexity of* DJAC *is $\Theta(D)$ where D is the diameter of JT.*

Proof: Let the degree of JT be bounded by K. Consider K time steps as one big time step. A constraint $r_i(R_i)$ will be "affected" by another constraint $r_j(R_j)$ iff given all the universal constraints along the path from j to i, the semijoin propagation from R_j to R_i is strictly included in R_i. 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. □

If the join tree is of unbounded degree, we can transform the join tree to a binary join tree which can be mapped to a distributed network. Furthermore, it is easy to see that if the join tree happens to be a balanced tree, then $D = O(\log n)$. However, in many cases, a join tree 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 treewidth, then we can find a balanced binary join tree, such that its acyclic constraint network, with size $poly(n)$ and bounded treewidth, is equivalent to ACN.

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

Proof: Let JT be the join tree of ACN and JT'' be the binary tree representation of JT and ACN'' be the acyclic constraint network of JT''. Let n'' and w'' be the size and treewidth 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 join tree T as input and returns the balanced binary join tree.

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 $JT' = BT(JT'', \emptyset, \emptyset)$ be the result of applying the above algorithm to JT''. Let ACN' be the acyclic constraint network of JT'. Since the height of JT' 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, treewidth(ACN') is bounded by $3w$. $\Box$

Since the diameter of the resultant join tree is $O(\log n)$, enforcing arc consistency in an acyclic constraint network of size n with bounded treewidth takes $O(\log n)$ time in a network of $poly(n)$ processors.

5 A Coarse-Grain Distributed Algorithm

In the previous two sections, we have presented the parallel `AC` algorithms for PRAM models and fine-grain interconnected network models. In both of these models, the number of processors is not bounded. However, most real parallel machines are coarse-grain interconnected networks, i.e. the number of processors is bounded by a constant. In this section, we give a distributed algorithm for arc consistency on join trees for coarse-grain interconnected network (CIN) models. Speedup and scaleup of the problem will be discussed for this class of models.

5.1 *The distributed algorithm* `DTAC`

The distributed algorithm `DTAC` is a distributed version of `TAC`, given a balanced binary join tree. Like the algorithms given by [CA91], `DTAC` distributes a problem into a set of processors statically. Let JT be a balanced binary join tree and let the processors be configured as a balanced binary tree PT. Let h_j and h_p be the height of JT and PT respectively, assuming $h_j \gg h_p$. There are three kinds of processors in PT: root processor (RP), internal processors (IPs) and leaf processors (LPs). We assign each subtree of JT rooted at depth h_p to each LP and assign the root and the rest of internal nodes of JT to the RP and IPs of PT respectively (Fig. 12).

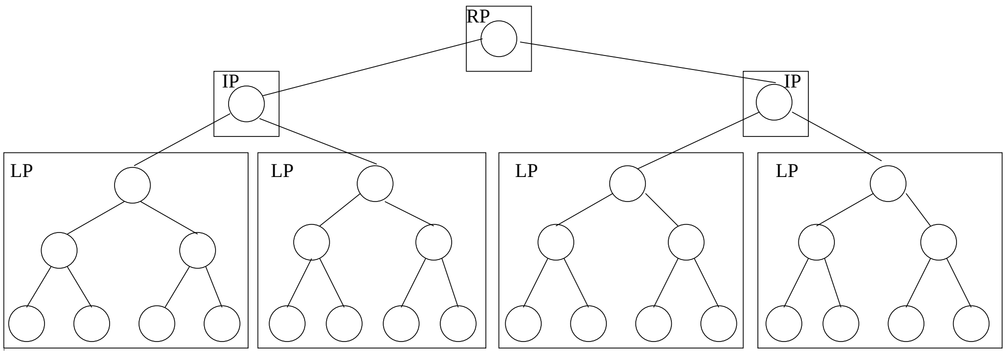

Figure 12: The problem distribution with $h_j = 4$ and $h_p = 2$

Algorithm `DTAC` consists of three types of processes: `RPAC`, `IPAC` and `LPAC`. `RPAC` (Fig. 13) is loaded on RP of PT. It combines the results from its children and sends the final result back. `IPAC` (figure 14) is loaded on each IP. It sends the result combined from its children to its parent and then sends the result combined with its parent back to its children. `LPAC` (Fig.

```
Algorithm RPAC: Root Algorithm
BEGIN   /* c(C) is the constraint on the node */
        receive l(L) from left;
        receive r(R) from right;
        c := c semijoin l;
        c := c semijoin r;
        send c(C) to left;
        send c(C) to right
END
```

Figure 13: The Root Processor algorithm RPAC

```
Algorithm IPAC: Internal Node Algorithm
BEGIN   /* c(C) is the constraint on the node */
        receive l(L) from left;
        receive r(R) from right;
        c := c semijoin l;
        c := c semijoin r;
        send c(C) to parent;
        receive p(P) from parent;
        c := c semijoin p;
        send c(C) to left;
        send c(C) to right
END
```

Figure 14: The Internal Processor algorithm IPAC

```
Algorithm LPAC: Leaf Node Algorithm
/* r(R) is the constraint on the subtree root */
BEGIN
        SemijoinUp(r); /* Fig. 16 */
        send r(R) to parent;
        receive p(P) from parent;
        r := r semijoin p;
        SemijoinDown(r); /* Fig. 16 */
END
```

Figure 15: The Leaf Processor algorithm LPAC

15) is loaded on each LP. It consists of two phases: the first phase combines the results up to the root of the subtree, the second phase propagates the final results back to each node in JT.

5.2 *Speedup and scaleup*

The ideal parallel system demonstrate two key properties [DG92]: (1) *linear speedup*: twice as much hardware can perform the task in half the elapsed time, and (2) *linear scaleup*: twice as much hardware can perform twice as large a task in the same elapsed time. Formally, the *speedup* of a system is measured as

$$speedup(N) = \frac{elapsed\_time\_using\_1\_processor}{elapsed\_time\_using\_N\_processors}.$$

The speedup is said to be *linear* if $speedup(N) = O(N)$. On the other hand, the *scaleup* of a system is measured as

$$scaleup(n, N) = \frac{elapsed\_time\_of\_size\_n\_problem\_using\_1\_processor}{elapsed\_time\_of\_size\_nN\_problem\_using\_N\_processors}.$$

The scaleup is said to be *linear* if $scaleup(n, N) = O(1)$.

```
Procedure SemijoinUp(c):
/* c(C) is the current constraint */
/* l(L) is the left constraint */
/* r(R) is the right constraint */
BEGIN
        IF (C is not a leaf node in JT) THEN
        BEGIN
                IF (there is left child l) THEN
                BEGIN
                        SemijoinUp(l);
                        c := c semijoin l
                END
                IF (there is right child r) THEN
                BEGIN
                        SemijoinUp(r);
                        c := c semijoin r;
                END
        END
END

Procedure SemijoinDown(c):
/* c(C) is the current constraint */
/* l(L) is the left constraint */
/* r(R) is the right constraint */
BEGIN
        IF (C is not a leaf node in JT) THEN
        BEGIN
                IF (there is left child l) THEN
                BEGIN
                        l := l semijoin c;
                        SemijoinDown(l)
                END
                IF (there is right child r) THEN
                BEGIN
                        r := r semijoin c;
                        SemijoinDown(r)
                END
        END
END
```


Figure 16: The procedures `SemijoinUp` and `SemijoinDown` for `LPAC`

In Section 4, we have shown that given a join tree of a constraint network with bounded treewidth, we can find an equivalent balanced binary join tree whose constraint network is also of bounded treewidth. In this section, we will show that given balanced binary join trees whose constraint networks are of bounded treewidth, `DTAC` has linear speedup and linear scaleup.

Theorem 5.1 *Given balanced binary join trees whose constraint networks are of bounded treewidth,* `DTAC` *has linear speedup and linear scaleup.*

Proof: let $N = 2^{h_p+1}-1$, $n = 2^{h_j+1}-1$. Given a tree-decomposition with bounded treewidth, the sequential complexity is linear in the number of tree nodes:

$$speedup(N) = \frac{O(2^{h_j+1}-1)}{O(2^{h_j-h_p+1}-1)+O(h_p)}.$$

Since $h_j \gg h_p$, $speedup(N) = O(2^{h_p}) = O(N)$. Thus it has linear speedup. Similarly,

$$scaleup(n, N) = \frac{O(2^{h_j+1}-1)}{O(2^{h_j+h_p-h_p+1}-1)+O(h_p)}.$$

We have $scaleup(n, N) = O(1)$. Thus it has linear scaleup. $\Box$

6 Experimental Results

Here we will present some experimental results on the algorithms given in the previous sections. We have developed a simulation environment for `DJAC` on FIN models based on logical time assumptions. In addition, we tested `DTAC` on a network of transputers. Both of these experiments are done on a set of generic constraint networks with a ring topology, two valued domain, and inequality relations. The size of the problem is the size of the ring minus one. The inputs to the algorithms are balanced binary join trees which are generated automatically (Fig. 17).

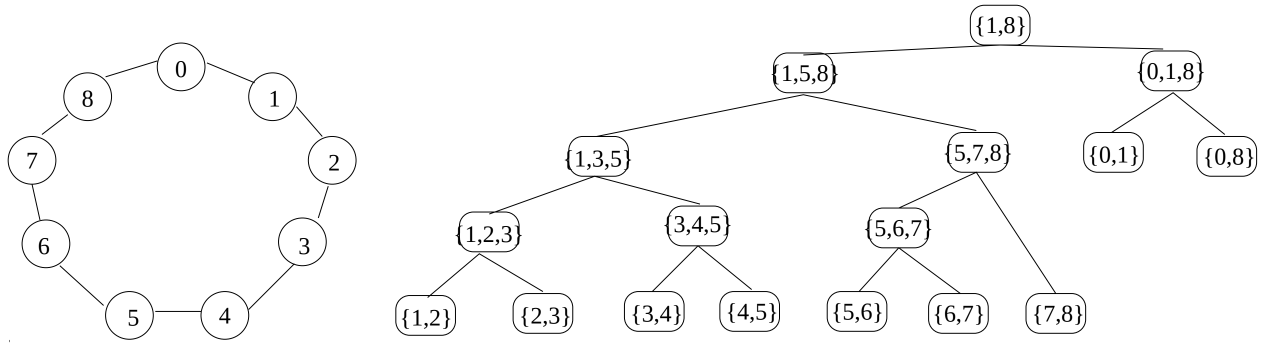

Figure 17: A ring structure and a balanced binary join tree

6.1 *Simulation of the distributed algorithm* `DJAC`

The simulation environment was developed in Strand88 [FT89]: a parallel logic programming language. Each node in the join network corresponds to a process, with the relation scheme as the identity, and with the local time and the relation tuples as the state. Each edge in the join network corresponds to a communication channel. Whenever a process receives a message from a neighbor process, consisting of the state of that process, it updates its state to a new state. The new relation tuples will be the result of the semijoin operation and the new local time will be the result of the logical time assumption defined as follows:

Definition 6.1 (The logical time assumption) *Each message of bounded size takes one unit time to arrive and each semijoin operation on relations of bounded size takes one unit time to compute. The time for sending and receiving messages is ignored.*

The logical time assumption is captured by the Strand88 program in Fig. 18.

If the relation tuples are refined because of the semijoin operation, the current state, the new relation tuples together with the new local time, is sent to all of its neighbor processes. The system will settle down to a stable state when no relation tuple is refined.

```
% T is the time when the message is sent
% Time is the local time
% CTime is the time when the message is arrived
% NewTime is the new local time

commTime(T, Time, CTime) :-
        T >= Time | CTime is T + 1.
commTime(T, Time, CTime) :-
        otherwise | CTime is Time.

newTime(T, Time, NewTime) :-
        commTime(T, Time, CTime),
        NewTime is CTime + 1.
```

Figure 18: The logical time assumption

In this environment, we simulated the distributed algorithm `DJAC` on a set of constraint networks with a ring topology, for the problem size of 4, 8, 16, 32 and 48. Figure 19 shows the time to stability vs. problem size on a log scale. The curve is almost linear which is consistent with the theoretical complexity.

6.2 *Performance of* `DTAC` *on a network of transputers*

We tested the algorithm `DTAC` on a network of transputers. Briefly, a transputer is a RISC-like microprocessor with four bidirectional bit serial links. The instruction set supports communication, concurrent processes and process scheduling. The hardware supports only nearest neighbor communication. The program is written in C++ and then compiled for the transputer. Table 2 summarizes the performance results.

The speedup for problem size 800 is shown in Fig. 20, The real performance is better because the memory access time increases rapidly with the increase of the problem size in

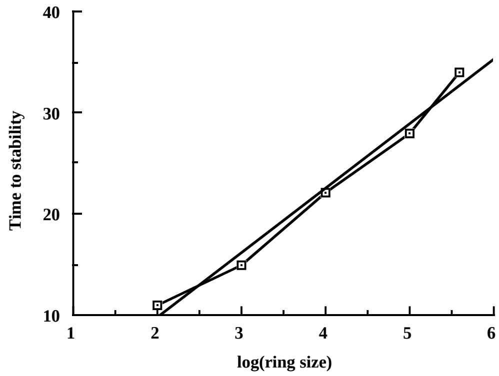

Figure 19: The time to stability vs. problem size on a log scale for `DJAC`

Table 2: The actual time performance of `DTAC`

Number of Processors	*Problem Size*						
	100	200	400	800	1600	3200	6400
1	0.25	0.71	2.29	8.02			
3		0.30	0.77	2.32	8.5		
7			0.39	0.86	2.46	8.17	
15				0.52	0.97	2.57	8.28

each processor. The scaleup for the problem is shown in Fig. 21, which demonstrates that

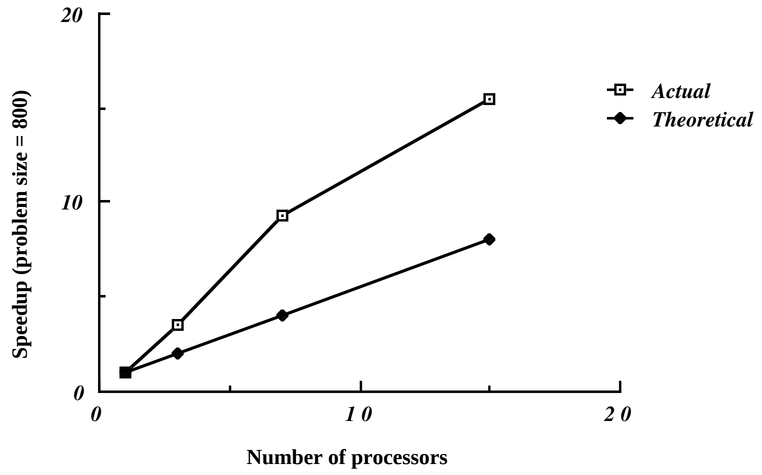

Figure 20: The speedup for `DTAC`

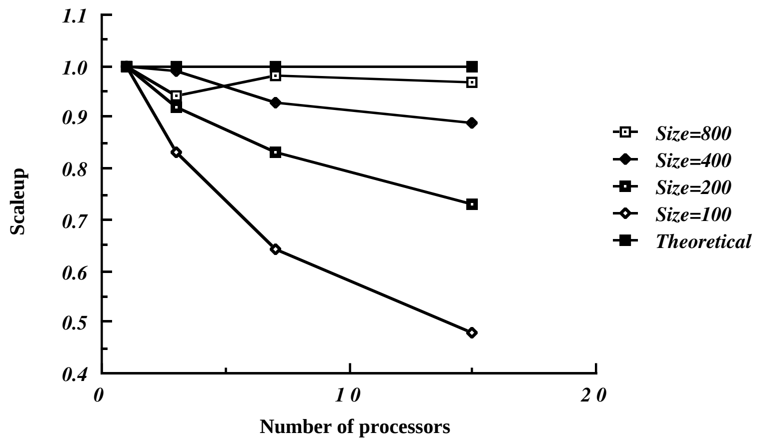

Figure 21: The scaleup for `DTAC`

the real scaleup is very close to linear, when the problem size is large in each processor.

7 Conclusions

We have presented three parallel `AC` algorithms for enforcing arc consistency in a join network. The FCSP decision problem can be reduced to the arc consistency problem if the tree-decomposition of the constraint network is given. The analysis shows that for an FCSP that can be represented by an acyclic constraint network of bounded treewidth, there are efficient algorithms in both parallel and distributed environments. The bounded treewidth property of constraint networks characterizes a set of tractable FCSPs as well as efficiently parallelizable FCSPs. The experimental results on simulation and real parallel machines show that good performance is achievable.

Acknowledgements

We wish to thank Rina Dechter, 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.

[AL91] S. Arnborg and J. Lagergren. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308 – 340, 1991.

[BFM+81] C. Beeri, R. Fagin, D. Maier, A. Mendelzon, J. Ullman, and M. Yannakakis. Properties of acyclic database schemes. In *ACM Symposium on Theory of Computing*, pages 335–362, 1981.

[Bod86] H. L. Bodlaender. Classes of graphs with bounded treewidth. Technical Report RUU-CS-86-22, Department of Computer Science, University of Utrecht, The Netherland, December 1986.

[Bod88] H. L. Bodlaender. NC-algorithms for graphs with small treewidth. Technical Report RUU-CS-88-4, Department of Computer Science, University of Utrecht, The Netherland, Februry 1988.

[CA91] J. M. Conrad and D. P. Agrawal. Performance of an asynchronous parallel algorithm on a generic multiprocessor simulator. In *Proceedings of the Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA, May 1991.

[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.

[Cou90] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science*, volume B, pages 193–242. The MIT Press/Elsevier, 1990.

[Dec92] R. Dechter. Constraint networks. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 276 – 285. Wiley, N.Y., 1992.

[DG92] D. DeWitt and J. Gray. Parallel database system: The future of high performance database systems. *Communication of ACM*, 35(6):85 – 98, June 1992.

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

[Fre82] E. C. Freuder. A sufficient condition for backtrack free search. *Journal of ACM*, 29(1):24 – 32, January 1982.

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

[FT89] I. Foster and S. Taylor. *Strand: New Concepts in Parallel Programming.* Prentice Hall, 1989.

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

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

[KR90] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science*, volume A, pages 869 – 941. The MIT Press/Elsevier, 1990.

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

[Mac92] A. K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 285 – 293. Wiley, N.Y., 1992.

[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.

[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.

[YDIK92] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalization distributed problem solving. In *Proceedings of the Twelfth International Conference on Distributed Systems*, pages 614 – 621, 1992.

[ZM91a] Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for constraint networks. Technical Report 91-6, Department of Computer Science, University of British Columbia, Vancouver, B.C. Canada, May 1991.

[ZM91b] Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *3rd IEEE Symposium on Parallel and Distributed Processing*, pages 394 – 397, Dallas, TX, December 1991.

[ZM91c] Y. Zhang and A. K. Mackworth. Parallel and distributed constraint satisfaction. In *Workshop on Parallel Processing in Artificial Intelligence*, pages 229 – 234, Sydney, Australia, August 1991.