Constraints, 5, 251–273 (2000)
© 2000 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

On Parallel Partial Solutions and Approximation Schemes for Local Consistency in Networks of Constraints

N. D. DENDRIS dendris@ceid.upatras.gr
Department of Computer Engineering and Informatics, Patras University, Rio, 265 00 Patras, Greece

L. M. KIROUSIS kirousis@ceid.upatras.gr
Department of Computer Engineering and Informatics, Patras University, Rio, 265 00 Patras, Greece

Y. C. STAMATIOU* stamatiu@ceid.upatras.gr
Department of Computer Engineering and Informatics, Patras University, Rio, 265 00 Patras, Greece

D. M. THILIKOS** sedthilk@ceid.upatras.gr
Department of Computer Engineering and Informatics, Patras University, Rio, 265 00 Patras, Greece

Abstract. A constraint network is arc consistent if any value of any of its variables is compatible with at least one value of any other variable. The Arc Consistency Problem (ACP) consists in filtering out values of the variables of a given network to obtain one that is arc consistent, without eliminating any solution. ACP is known to be inherently sequential, or P-complete, so in this paper we examine some weaker versions of it and their parallel complexity. We propose several natural approximation schemes for ACP and show that they are also P-complete. In an attempt to overcome these negative results, we turn our attention to the problem of filtering out values from the variables so that each value in the resulting network is compatible with at least one value of not necessarily all, but a constant fraction of the other variables. We call such a network partially arc consistent. We give a parallel algorithm that, for any constraint network, outputs a partially arc consistent subnetwork of it in sublinear ($O(\sqrt{n}\log n)$) parallel time using $O(n^2)$ processors. This is the first (to our knowledge) sublinear-time parallel algorithm with polynomially many processors that guarantees that in the resulting network every value is compatible with at least one value in at least a constant fraction of the remaining variables. Finally, we generalize the notion of partiality to the k-consistency problem.

Keywords: constraint satisfaction problem, arc consistency, local consistency, approximation schemes, partial arc consistency, P-completeness, parallel algorithms

1. Introduction

A *constraint network* comprises of n variables, a *domain* of permissible values for each of the variables, and a set of *constraint relations*, each binding the values of a collection of the variables. The Constraint Satisfaction Problem (CSP) is the problem of determining the n-tuples of values from the domains that are compatible with all constraints. This is

* Partially supported by the General Secretariat of Research and Technology of Greece through the research project YPER 1248. Also supported by European Union projects ALCOM II (contract no. 7141) and ALCOM IT (contract no. 20244).
** Partially supported by the European Union's projects ALCOM II (contract no. 7141), GEPPCOM (contract no. 9072), INSIGHT II (contract no. 6019), and ALCOM IT (contract no. 20244), by the Training and Mobility of Researchers (TMR) Program, (EU contract no ERBFMBICT950198), by the NSERC of Canada, and by the Ministry of Education and Culture of Spain – Grant no. MEC-DGES SB97 0K148809.

a fundamental problem in Artificial Intelligence (an interesting special case of it is scene labeling, a preprocessing stage of object recognition). For a review of recent results see [6], [21], [23]. CSP is, in general, intractable. Many NP-complete problems such as graph colouring, 3-SAT, can be directly expressed in terms of CSP. There is an extensive literature on special cases of CSP that can be solved efficiently sequentially, or even by fast parallel algorithms (see, e.g., [3], [8], [12], [19], [26]).

A constraint network is called arc consistent if any value from any domain is compatible with at least one value of any other variable. *Discrete relaxation* is the process of removing values from the domains of the variables until the network is transformed to an arc consistent one, but in a way that no n-tuple of values that satisfies all constraints is eliminated. In other words, discrete relaxation solves the Arc Consistency Problem (ACP). It removes local inconsistencies and so it reduces the search space. Therefore, a network that has been transformed to an arc consistent one is expected to be easier to handle from the point of view of obtaining a solution for the corresponding CSP.

There are many efficient sequential implementations of discrete relaxation (see [22]; see also [15] for an overview and recent results and see [7] for interesting experimental results). Parallel algorithms for the arc consistency problem have been studied in [5], [24]. However, in general, the problem is inherently sequential, i.e. it is P-complete [17]. In Section 4 we prove a stronger version of this result. There are also interesting results on identifying cases where an exponential parallel speedup is possible (see [18]).

In this paper, in order to overcome the inherent sequentiality of the problem, we introduce and investigate the parallel complexity of partial solutions for ACP. We justify this approach by defining several natural approximation schemes to the problem of arc consistency, and showing them to be P-complete. We also extend the notion of partiality to cover the case of the more general k-consistency.

An α-partial solution of ACP is one where every value of any variable is compatible with values of at least $\alpha(n-1)$ of the other variables ($0 < \alpha \leq 1$). Let Δ be the maximum degree in the graph of constraints, and q be the maximum number of values for a variable. Usually, q is a small constant. For example, in many applications of CSP in Computer Vision, q is no more than 3 or 4 (see [25]; see [9] for some recent results). Also, in the majority of cases, Δ is a constant factor of the number of variables. We show that, for any $\alpha < 1 - \frac{q}{q+1}\frac{\Delta}{n-1}$, there is a parallel algorithm that runs in time $O(\sqrt{nq}\log(nq))$ and produces an α-partial solution for ACP that contains the actual ACP solution.

Our algorithm operates as follows: For as long as there are more than $\sqrt{nq}$ values that cannot be part of the ACP solution (i.e. they are compatible with values of less than $n-1$ variables), the algorithm removes them in parallel steps. The target is to capture the steps of the original discrete relaxation that offer sufficiently high degree of parallelism. Afterwards, the algorithm removes values that are compatible with less than $\alpha(n-1)$ variables. A combinatorial lemma ensures that these steps will not be more than $O(\sqrt{nq})$. The algorithm requires $O(n^2q^2)$ processors on the Concurrent-Read Exclusive-Write (CREW) shared-memory parallel machine model (PRAM).

Notice that a partial solution for ACP is, in itself, a removal of some local inconsistencies. To our knowledge, this is the first sublinear-time parallel algorithm that achieves a nontrivial degree of elimination of local inconsistencies, using polynomially many processors (see [5] for results with exponentially many processors).

We next examine several approximation schemes to ACP. These approximations are all proved to be P-complete. To begin with, let D be the disjoint union of all domains of a network, D_{AC} be the disjoint union of all domains *after* the application of discrete relaxation, and $R_{\text{AC}} = D - D_{\text{AC}}$. Our first approximation scheme asks to determine, for a given $\epsilon \in (0, 1]$, a set D' such that $D_{\text{AC}} \subseteq D' \subseteq D$ and $\epsilon|D'| \leq| D_{\text{AC}}|$. We prove that, unless P=NC, no such set D' can be found by a parallel algorithm in NC (for the class NC see [16]). We show the same for the approximation scheme that, given an $\alpha \in (0, 1]$, asks for a set R' such that $R' \subseteq R_{\text{AC}}$ and $\alpha|R_{\text{AC}}| \leq| R'|$. The last approximation scheme we introduce is related to the degree of incompatibility that a value must exhibit in order to be removed by a relaxation-type algorithm. We define the elimination degree of d as the least k such that, when we remove values not supported by more than k variables, d is not removed and no domain is emptied (see next sections for formal definitions). We show that approximating this parameter within *any* approximation factor, is P-complete. Our conclusion is that looking for partial solutions, and not for approximation schemes, seems to be the correct way to obtain sublinear-time parallel algorithms that approach the solution of ACP.

To this extent, in the last section of the paper, we generalize our partiality results to the case of k-consistency, a generalized form of arc consistency, as well as for the case of strong k-consistency. A constraint network is k-consistent if for any $(k - 1)$-tuple of compatible values for $k-1$ variables and for any kth variable, there exists a value for it that is compatible with the $(k-1)$-tuple. A network which is i-consistent for all $i = 2, \ldots, k$, is called strongly k-consistent (see [6], [21]). k-consistency can be achieved in time $O(n^k q^k)$ [4].

For k-consistency, we extend the notion of partiality as follows: a network is α-partially k-consistent if any $(k - 1)$-tuple of compatible values can be extended with a compatible value from the domain of not necessarily all, but at least $\alpha(n - k + 1)$ other variables. If a network has the α-partial k-consistency property for all $i = 2, \ldots, k$, then we call it α-partially strongly k-consistent. We give a parallel algorithm for the CREW PRAM that, in time $O((nq)^{\frac{k-1}{2}} \log(nq))$ and using $O(n^k q^k)$ processors, outputs an α-partially strongly k-consistent subnetwork of any given constraint network that contains the solution of the strong k-consistency problem.

2. Definitions and Preliminaries

A *binary constraint network* consists of a set of variables $X_1, \ldots, X_n$, a set of variable domains $D_1, \ldots, D_n$, and a set of binary constraints $\mathcal{C}$. For $i = 1, \ldots, n$, D_i is the set of permissible values for variable X_i. A constraint $R_{ij} \in \mathcal{C}$ is a subset of $D_i \times D_j$ $(i \neq j)$. A value d_i of X_i is *compatible* with a value d_j of X_j $(i \neq j)$ iff there is no $R_{ij} \in \mathcal{C}$ such that $(d_i, d_j) \notin R_{ij}$. For a given constraint network $\mathcal{N}$ with variables $X_1, \ldots, X_n$, domains $D_1, \ldots, D_n$, and set of constraints $\mathcal{C}$, a *domain-reduced subnetwork* of $\mathcal{N}$ is any constraint network $\mathcal{N}'$ with the same variables, domains $D'_1 \subseteq D_1, \ldots, D'_n \subseteq D_n$, and set of constraints $\mathcal{C}'$ containing exactly the restrictions of the elements of $\mathcal{C}$ to the corresponding domains.

The *Constraint Satisfaction Problem* (CSP), for a given constraint network, asks for all the n-tuples $(d_1, \dots, d_n)$ of values such that $d_i \in D_i, i = 1, \dots, n$, and for every $R_{ij} \in \mathcal{C}$, (d_i, d_j) is in R_{ij}. Such an n-tuple is called a *solution* of the CSP. The decision version of the CSP is to determine if any solution exists.

Given an instance Π of the CSP with n variables, its *constraint graph* (we denote this graph by G^{Π} or just G when no confusion may arise) has n vertices which correspond to the variables of Π and it contains an edge $\{v_i, v_j\}$ iff the corresponding variables are bound by a constraint.

In this paper we use another graph representation of a CSP instance Π, which we call the *compatibility graph*. Let q denote the maximum number of values that can be assigned to a variable, i.e. $q = \max\{|D_i|, i = 1, \dots, n\}$. The compatibility graph C^{Π} (or just C, when no confusion may arise) of Π is an n-partite graph. The ith part of C^{Π} corresponds to variable X_i of Π and it has exactly $|D_i|$ vertices, one for each value of X_i. In the compatibility graph C^{Π}, two vertices u and v are connected by an edge iff the corresponding values $d_i \in D_i$ and $d_j \in D_j$ are compatible. We denote by N the total number of vertices of C^{Π}, i.e. $N = |D|$ where $D = \bigcup_{i=1,\dots,n} D_i$ (note that $N \leq nq$). In other words, N is equal to the total number of values in the constraint network (values in the domains of different variables are considered to be different). An n-tuple $(d_1, \dots, d_n)$ is a solution of Π iff the set of vertices that correspond to the values in the n-tuple induces a subgraph of C^{Π} which is an n-clique. Therefore, the problem of finding all the solutions of a CSP instance, or of determining whether any solution at all exists, can be reduced to the problem of identifying *all* n-cliques of its compatibility graph, or of determining whether at least one such clique exists.

In order to avoid confusion, the following point should be clarified: whenever two variables X_i and X_j of a constraint network are not bound by a constraint, i.e. the corresponding vertices are not adjacent in G^{Π}, then the vertices of the part in C^{Π} that corresponds to X_i are adjacent to all the vertices of the part of C^{Π} that corresponds to X_j. However, if we are given a compatibility graph C^{Π} and all the pairs of vertices from two of its parts are adjacent, then we do not consider that the corresponding to the variables vertices in G^{Π} are adjacent. That is, we do not assume that the variables are bound by a constraint that simply allows any combination of value assignments to the variables. In this way, given a constraint graph C, there is a unique compatibility graph G associated with C.

A constraint network $\mathcal{N}$ is *arc consistent* if the following holds: for any variable X_i, for any value $d_i \in D_i$, and for any other variable X_j, there exists at least one value assignment $d_j \in D_j$ such that d_i is compatible with d_j. The *Arc Consistency Problem* (ACP) is the problem of finding a maximal (with respect to the domains) arc consistent domain-reduced subnetwork $\mathcal{N}_{\text{AC}}$ of $\mathcal{N}$. Also, $\mathcal{N}_{\text{AC}}$ is unique, for if there was another maximal arc consistent subnetwork $\mathcal{N}'_{\text{AC}}$ of $\mathcal{N}$ with domains $A'_1, \dots, A'_n$, then the domain-reduced subnetwork of $\mathcal{N}$ with domains $A_1 \cup A'_1, \dots, A_n \cup A'_n$ would also be arc consistent, a contradiction to the maximality of $\mathcal{N}_{\text{AC}}$ and $\mathcal{N}'_{\text{AC}}$. We call $\mathcal{N}_{\text{AC}}$ the *solution* of ACP.

In terms of the compatibility graph C, ACP is formulated as follows: Find the maximal (with respect to the set of vertices) induced subgraph C_{AC} of C such that any vertex in any of its parts is connected with vertices in the $n - 1$ other parts.

All values $d_i \in D_i, i = 1, \dots, n$, which participate in some solution of the CSP belong to the domain A_i of the ACP solution. Since CSP is an NP-complete problem, a common approach for solving it involves a preprocessing step to make the network arc consistent and reduce the size of the variable domains, before employing exhaustive search to determine the actual solutions. Arc consistency is achieved by the procedure known as *discrete relaxation*. The algorithm is the following:

```
Algorithm: discrete relaxation
Input:     A constraint network $\mathcal{N}$
Output:    $\mathcal{N}_{AC}$
1. begin
2.    while $\exists X_i$ and $d_i \in D_i$: $\exists j \neq i$: $d_i$ is incompatible with all $d_j \in D_j$
3.    do $D_i \leftarrow D_i - \{d_i\}$ od
4. end
```

The algorithm removes these values, if any, that are incompatible with all the values in the domain of at least one variable, until no such values exist. Discrete relaxation runs in optimal $O(eq^2)$ time [2], where e is the number of constraints and q is the maximum number of values in the domain of a variable. Also, several parallel implementations of the discrete relaxation for the PRAM model of computation are described in [24]. The best of these algorithms, PAC-4, executes in $O(nq)$ time using $O(n^2q^2)$ processors. A work optimal parallel algorithm for the CRCW PRAM model that enforcess arc-consistency in $O(nq)$ time using $O(nq)$ processors, is described in [18].

3. Partial Arc Consistency

In this section we introduce the notion of partiality in arc consistency, and give a sublinear-time parallel algorithm that finds partial solutions to ACP (for other relevant combinatorial problems that admit partial solutions see [20]).

A constraint network $\mathcal{N}$ is *α-partially arc consistent* if no variable domain is empty and any value of a variable is compatible with values of at least a constant fraction of the other variables, i.e. with values of $\alpha(n-1)$ other variables ($0 < \alpha \leq 1$). The *α-partial arc consistency problem* is the problem of finding a maximal (with respect to the domains) α-partially arc consistent domain-reduced subnetwork $\mathcal{N}_{AC,\alpha}$ of $\mathcal{N}$. When $\alpha = 1$, this problem is the same as the original ACP. In a way similar to the case of $\mathcal{N}_{AC}$, one can see that $\mathcal{N}_{AC,\alpha}$ is unique.

For any value d in a constraint network, the *support-degree* of d is the number of variables in the network that contain a value compatible with d. In the compatibility graph C, the support-degree of a vertex v in C is the number of parts in C that contain vertices adjacent to v.

In terms of the compatibility graph C, the α-partial arc consistency problem is formulated as follows: Find the maximal (with respect to the set of vertices) induced subgraph $C_{AC,\alpha}$ of C such that any vertex has support-degree at least $\alpha(n-1)$ and no part is empty, i.e. each part has at least one vertex. Such a graph is called α-partially arc consistent.

In this paper we introduce a modified version of the discrete relaxation algorithm, which we call *α-discrete relaxation*, that solves the α-partial arc consistency problem. We will present the algorithm in terms of the compatibility graph. Given a real value α, $0 < \alpha \leq 1$

and a compatibility graph C, the α-discrete relaxation algorithm is the following:

```
Algorithm: $\alpha$-discrete relaxation
Input:     An $n$-partite compatibility graph $C = (V, E)$
Output:    $C_{\text{AC},\alpha}$
1. begin
2.    while $\exists v \in V$ : support-degree($v$) $< \alpha(n-1)$
3.    do
4.       $V \leftarrow V - \{v\}$
5.       if an empty part appears in $C$
6.       then terminate and return the empty graph
7.    od
8. end
```

It is easy to see that our algorithm outputs $C_{\text{AC},\alpha}$. Notice that $C_{\text{AC},\alpha}$ contains C_{AC} as a subgraph. Also notice that if during the α-discrete relaxation a part is emptied, then the ACP has the empty solution.

The algorithm α-discrete relaxation has a parallel version which we call *parallel α-discrete relaxation*. In this version, on each step of the while loop the identification and the elimination of the vertices to be removed is done in parallel in $O(\log(nq))$ time using $O(n^2q^2)$ processors on the CREW PRAM. Therefore, the α-partial arc consistency problem can be solved in $O(nq\log(nq))$ parallel time using $O(n^2q^2)$ processors on the CREW PRAM.

A key observation regarding partial arc consistency is the following: The *maximality* of an α-partially arc consistent subgraph of C is not crucial, as long as this subgraph contains C_{AC}. It turns out that we can do better than computing $C_{\text{AC},\alpha}$. That is, given a constraint network $\mathcal{N}$, its constraint graph C and its compatibility graph G, and for any $\alpha < 1 - \frac{q}{q+1}\frac{\Delta}{n-1}$, we can compute in sublinear parallel time an α-partially arc consistent subgraph of $C_{\text{AC},\alpha}$ that contains C_{AC}. In the above, $\Delta = \Delta(G)$ is the maximum degree in the constraint graph G.

The algorithm is strongly based on the following lemma. This lemma ensures the existence of a partially consistent subgraph of C whenever C is sufficiently dense.

Lemma 1 *Let $\mathcal{N}$ be a constraint network with n variables each one having at most q values. Let G, C be its constraint and compatibility graph respectively, Δ the maximum degree of G, and l the smallest possible support-degree of any value in $\mathcal{N}$ (observe that $l = n - 1 - \Delta$). Also let $l_2 < l_1 \leq \Delta$ be two integers such that $l_2 < \frac{l_1}{1+q}$. If C contains at most m vertices with support-degree less than $l + l_1$, then, if we apply $(\frac{l+l_2}{n-1})$-discrete relaxation on C, either no more than $\frac{m(l_1-l_2)}{l_1-l_2(1+q)}$ vertices will be removed, or all the vertices of some part will be removed.*

Proof: From the definition of the compatibility graph, any vertex v in C is adjacent to *at least* one vertex of *at least* l parts, i.e. v has support-degree at least l. Therefore, applying an $(\frac{l'}{n-1})$-discrete relaxation has no effect on the graph C, for $l' \leq l$. Let us consider the application of an $(\frac{l+l_2}{n-1})$-discrete relaxation on C for $l_2 \geq 0$, and suppose that no part is emptied. Suppose, also, that we are given another positive integer l_1 such that $l_2 < l_1$ (we will discover later the exact relation between l_1 and l_2). The application of an $(\frac{l+l_2}{n-1})$-discrete relaxation on C removes a set R of vertices in C, which can be partitioned into the following sets:

- The set R_1: all vertices that initially (in C) have support-degree less than $l + l_2$.
- The set R_2: all vertices that initially have support-degree no less than $l + l_2$ but less than $l + l_1$.
- The set R_3: all vertices that initially have support-degree at least $l + l_1$.

We will determine an upper bound to the size of R, which, in turn, will enable us to bound the number of steps required by the $(\frac{l+l_2}{n-1})$-discrete relaxation. We will count the number of elements in R indirectly, by considering the set E_R of edges that are removed as a result of the removal of the vertices. In what follows, we define as var(v) the vertex of the constraint graph corresponding to the part of C that contains v. A crucial observation resulting from our assumption that no part in C is emptied, is that for any vertex v in C, v can have its support-degree reduced only as a result of the removal of a vertex v' such that var(v) and var(v') are adjacent in the constraint graph G. We call such a vertex v' a *direct* neighbour of v.

Let v be a vertex in R_3. Since the vertices in R_3 have initially support-degree at least $l+l_1$, and we apply an $(\frac{l+l_2}{n-1})$-discrete relaxation with $l_2 < l_1$, v must have lost at least $l_1 - l_2$ of its direct neighbours in C by the time it is removed. Therefore, at least $|R_3|(l_1 - l_2)$ edges must have been removed as a result of the removal of the vertices in R_3, i.e.:

$$|R_3|(l_1 - l_2) \leq |E_R| \tag{1}$$

Let v be a vertex in R. Then v cannot be adjacent to more than ql_2 direct neighbours in C by the time it is removed. At worst, all these direct neighbours will be subsequently removed. Therefore $|E_R| \leq| R|ql_2$, i.e. at most $|R|ql_2$ edges are removed in all. From our assumption that at most m vertices in C have support-degree less than $l + l_1$, it follows that $|R_1| + |R_2| \leq m$. Since $R = R_1 \cup R_2 \cup R_3$, we get that $|R| \leq m + |R_3|$. Thus:

$$E_R \leq |R|ql_2 \leq (m + |R_3|)ql_2 \tag{2}$$

From 1 and 2:

$$\begin{aligned}
(l_1 - l_2)|R_3| &\leq (m + |R_3|)ql_2 \Rightarrow \\
(l_1 - l_2(q + 1))|R_3| &\leq ql_2 m \Rightarrow \\
|R_3| &\leq \frac{ql_2 m}{l_1 - l_2(q + 1)}
\end{aligned}$$

Since $|R| \leq (m + |R_3|)$, it follows that:

$$\begin{aligned}
|R| &\leq m + \frac{ql_2 m}{l_1 - l_2(q + 1)} \Rightarrow \\
|R| &\leq m\frac{l_1 - l_2}{l_1 - l_2(q + 1)}
\end{aligned}$$

Now, we can see that l_1 and l_2 must be chosen so that $l_2 < \frac{l_1}{q+1}$. ■

The lemma implies that if the the number of vertices in C that have low degree is sufficiently small, i.e. C is sufficiently dense, then we can find a bound on the number of vertices that will be removed by the α-discrete relaxation. This, in turn, suggests the following idea: as long as the number of vertices with support-degree is above a threshold, remove them in parallel. When their number (the value m in Lemma 1) falls below this threshold, apply parallel α-discrete relaxation. If we arrange things so that the threshold is sufficiently low, then, from Lemma 1, the application of the parallel α-discrete relaxation (for an appropriate value of α, of course) will remove a bounded number of vertices, thus its worst case running time will be bounded by the same bound. All these considerations are formalized by the following theorem:

Theorem 1 *Let $\mathcal{N}$ be a constraint network with n variables each one having at most q values. Let also G, C be its constraint and compatibility graph respectively. If G has maximum degree Δ, then for any $\alpha < 1 - \frac{q}{q+1}\frac{\Delta}{n-1}$, there exists a parallel algorithm that returns a subgraph of $C_{\mathrm{AC},\alpha}$ which contains C_{AC} and where any vertex has support-degree at least $\alpha(n-1)$. This algorithm executes in $O(\sqrt{nq}\log(nq))$ time and uses $O(n^2q^2)$ processors on the CREW PRAM.*

Proof: We present the algorithm in terms of the compatibility graph:

```
Algorithm: fast $\alpha$-discrete relaxation
Input:     An $n$-partite compatibility graph $C = (V, E)$
Output:    An $\alpha$-partially arc consistent subgraph $C'$ of $C_{\mathrm{AC},\alpha}$ that
           contains $C_{\mathrm{AC}}$, where $\alpha < 1 - \frac{q}{q+1}\frac{\Delta}{n-1}$
1.  begin
2.    $N \leftarrow |V|$
3.    while $\exists\, V' \subseteq V : (|V'| \geq \sqrt{N}) \wedge (\forall v \in V' : \text{support-degree}(v) < n - 1)$
4.    do
5.      for each $v \in V'$ in parallel
6.      do $V \leftarrow V - \{v\}$ od
7.      if an empty part appears in $C$
9.      then terminate and return the empty graph
10.   od
11.   run parallel $\alpha$-discrete relaxation
12. end
```

It is easy to see that each vertex in the subgraph the algorithm outputs has support-degree no less than $\alpha(n-1)$.

For the time and processor bounds, we observe that the identification of each set V' can be performed in $O(\log N)$ parallel time, using $O(N^2)$ processors on a CREW PRAM. In each such step, the algorithm removes (in constant time) at least $\sqrt{N}$ vertices inside the parallel for-loop. Therefore, the while-loop has at most $\sqrt{N}$ iterations. Also, after the removal of V', checking whether any part of C is empty requires $O(\log N)$ parallel time. Summing up, the algorithm exits the while-loop in time $O(\sqrt{N}\log N)$. Also, after the end of the while loop, there are less than $\sqrt{N}$ vertices in V that have support-degree less than

$n-1$. Then, by Lemma 1 and since $l_2 + l = \alpha(n-1) \Rightarrow l_2 = \alpha(n-1) - l$ we get that α-discrete relaxation will remove no more than

$$\sqrt{N}\frac{l_1 - (\alpha(n-1)+l)}{l_1 - (q+1)(a(n-1)-l)} \tag{3}$$

vertices.

However, if we wish Lemma 1 to apply, we must obey the inequality $l_2 < \frac{l_1}{q+1}$ and, thus, confine α to the following range:

$$\begin{aligned} l_1 &> (q+1)l_2 \Rightarrow \\ l_1 &> (q+1)(\alpha(n-1)-l) \Rightarrow \\ &\vdots \\ \alpha &< 1 - \frac{1}{n-1}(\Delta - \frac{l_1}{q+1}) \end{aligned}$$

Observe that the bigger α is, the better the partial solution we get. It is clear that the allowed range for α is increased as l_1 is increased. Therefore, we must set l_1 to the maximum possible value. Since $l_1 + l$ signifies support-degree, the following must hold:

$$\begin{aligned} l_1 + l \leq n-1 &\Rightarrow \\ l_1 + n - 1 - \Delta \leq n-1 &\Rightarrow l_1 \leq \Delta \end{aligned}$$

By setting $l_1 = \Delta$ we get the maximum possible range for α and, from Equation 3, we conclude that α-discrete relaxation will remove no more than

$$\sqrt{N}\frac{1-\alpha}{(1-\alpha)(1+q) - q\frac{\Delta}{n-1}}$$

vertices. This number, for the range of α stated above, is $O(\sqrt{N})$. Thus, parallel α-discrete relaxation will execute in $O(\sqrt{N}\log N)$ parallel time, using $O(N^2)$ processors on the CREW PRAM. It follows that fast α-discrete relaxation executes in $O(\sqrt{N}\log N)$ time using $O(N^2)$ processors. Since $N \leq nq$, we have the required time and processor bounds for α-partial AC. ■

4. Parallel Intractability Results

In this section, we first give a P-completeness result concerning ACP. Subsequently, we present several natural approximation schemes for ACP, which we prove to be P-complete.

As mentioned in the introduction, Kasif [17] proved that the arc consistency problem is P-complete, in the sense that given a constraint network $\mathcal{N}$, it is P-complete to decide whether a given value for a variable belongs in the corresponding domain of $\mathcal{N}_{\text{AC}}$. Here, we prove the following stronger result (the proof of all the Theorems of this section are given in the Appendix):

Theorem 2 *It is P-complete to decide whether discrete relaxation in a network $\mathcal{N}$ removes a specific value of one of its variables, or if it empties the domains of all of its variables, even if the constraint graph of $\mathcal{N}$ is planar and has maximum degree 3.*

We will now examine two domain approximation schemes. Let $\mathcal{N}$ be a constraint network with n variables and variable domains $D_1, \ldots, D_n$. Let also $A_1, \ldots, A_n$ denote the domains in $\mathcal{N}_{\text{AC}}$. If $D = \bigcup_i \{D_i, i = 1, \ldots, n\}$, $D_{\text{AC}} = \bigcup_i \{A_i, i = 1, \ldots, n\}$, and $R_{\text{AC}} = D - D_{\text{AC}}$, then the following hold:

Theorem 3 *For a given constraint network $\mathcal{N}$, and for any $\epsilon \in (0, 1]$, it is P-complete to find a set D' such that $D_{\text{AC}} \subseteq D' \subseteq D$ and $\epsilon|D'| \leq| D_{\text{AC}}|$.*

Theorem 4 *For a given constraint network $\mathcal{N}$, and for any $\epsilon \in (0, 1]$, it is P-complete to find a set R' such that $R' \subseteq R_{\text{AC}}$ and $\epsilon|R_{\text{AC}}| \leq| R'|$.*

We now define the *elimination degree* of value d of $\mathcal{N}$ to be the least integer h such that a $(\frac{n-h}{n-1})$-discrete relaxation does not remove d. We denote this quantity by $\text{elimin}(d, \mathcal{N})$. It is easy to see that:

$$\text{support}(d, \mathcal{N}) + \text{elimin}(d, \mathcal{N}) = n$$

It follows that it is P-complete to find the value of $\text{elimin}(d, \mathcal{N})$. Since the computation of the elimination degree of d is a minimization problem, we consider the following approximation for it: given a constant factor $\lambda \geq 1$, find an integer $\text{elimin}(d, \mathcal{N})_{\text{approx}}$ such that:

$$\lambda \cdot \text{elimin}(d, \mathcal{N}) \geq \text{elimin}(d, \mathcal{N})_{\text{approx}} \geq \text{elimin}(d, \mathcal{N})$$

Theorem 5 *It is P-complete to approximate* $\text{elimin}(d, \mathcal{N})$ *by any factor* $\lambda \geq 1$.

5. Extensions for Arbitrary Degree of Consistency

We will now extend our results to the case of k-consistency, both for CSPs with binary constraints, and for constraint networks where the relations are not necessarily binary. In the latter case, a *constraint network* consists of a set of variables $X_1, \ldots, X_n$, a set of variable domains $D_1, \ldots, D_n$, and a set of constraints $\mathcal{C}$. Let $I = \{1, \ldots, n\}$ and $I_r = \{i_1, \ldots, i_r\} \subseteq I$ such that $j < j' \Rightarrow i_j < i_{j'}$. A constraint $R_{I_r} \in \mathcal{C}$ of *arity* r is a subset of $D_{i_1} \times \cdots \times D_{i_r}$. The *arity* of $\mathcal{N}$ is equal to the maximum arity among its constraints. Also, a set of value assignments $\{d_{i_1}, \ldots, d_{i_r}\}$ where $d_{i_j} \in X_{i_j}$, $j = 1, \ldots, r$, is *compatible* iff there is no $R_{I_r} \in \mathcal{C}$ such that $(d_{i_1}, \ldots, d_{i_r}) \notin R_{I_r}$.

The *Constraint Satisfaction Problem* (CSP) for the general case, asks for all the n-tuples $(d_1, \ldots, d_n)$ of values such that $d_i \in D_i$, $i = 1, \ldots, n$, and for every $R_{I_r} \in \mathcal{C}$, $(d_{i_1}, \ldots, d_{i_r}) \in R_{I_r}$. Such an n-tuple is called a *solution* of the CSP.

Similarly with the notion of the constraint graph, for a CSP instance Π with n variables, its *constraint hypergraph* G^{Π} has n vertices corresponding to the variables of Π. G^{Π} contains a hyperedge iff the variables that correspond to the vertices of the hyperedge are bound by a constraint. Also, the *compatibility hypergraph* C^{Π} of Π has exactly one vertex for each value in the domain of each variable. A set of vertices $\{v_{i_1}, \ldots, v_{i_r}\}, v_{i_j} \in V_{i_j}, j = 1, \ldots, r$, is a hyperedge in C^{Π} iff the set of values that corresponds to $\{v_{i_1}, \ldots, v_{i_r}\}$ is compatible. An n-tuple $(d_1, \ldots, d_n)$ of values is a solution of Π iff any subset of the corresponding vertex set of C^{Π} that has cardinality ≥ 2 is a hyperedge.

A constraint network $\mathcal{N}$ is *k-consistent* if the following holds: for any set of value assignments $\{d_{i_1}, \ldots, d_{i_{k-1}}\}$ such that every subset of it is compatible, and for any $i \notin \{i_1, \ldots, i_{k-1}\}$, there exists a value assignment $d_i \in D_i$ such that every subset of $\{d_{i_1}, \ldots, d_{i_{k-1}}\} \cup \{d_i\}$ is compatible. The *k-consistency problem* is the problem of finding a maximal (with respect to the domains) k-consistent subnetwork $\mathcal{N}_k$ of $\mathcal{N}$. It is easy to see that $\mathcal{N}_k$ is unique. We call $\mathcal{N}_k$ the *solution* of k-consistency.

A constraint network $\mathcal{N}$ is *strongly k-consistent* if it is i-consistent for any $i, i = 2, \ldots, k$. The *strong k-consistency problem* is the problem of finding a maximal (with respect to the domains) strongly k-consistent subnetwork $\mathcal{N}_k^s$ of $\mathcal{N}$.

A hyperedge $\{v_{i_1}, \ldots, v_{i_r}\}, r \geq 2$, of a compatibility hypergraph C is a *hyperclique* iff any subset of $\{v_{i_1}, \ldots, v_{i_r}\}$ is a hyperedge. We call such a hyperclique an *r-hyperclique*; also, any vertex v in C is considered to be an 1-hyperclique. The *support-degree* of an r-hyperclique $\{v_{i_1}, \ldots, v_{i_r}\}$ is the number of parts not in $\{V_{i_1}, \ldots, V_{i_r}\}$ that contain a vertex v such that $\{v_{i_1}, \ldots, v_{i_r}, v\}$ is an $(r + 1)$-hyperclique.

In terms of the compatibility hypergraph, the k-consistency problem is formulated as follows: Find its maximal (with respect to the set of vertices) induced subhypergraph C_k in which any $(k - 1)$-hyperclique has support-degree equal to $n - k + 1$ (we call such a hypergraph *k-consistent*). The *strong k-consistency problem* is the problem of finding a maximal (with respect to the set of vertices) induced subhypergraph C_k^s that is i-consistent for all $i, i = 2, \ldots, k$.

k-consistency is achieved by the procedure called *k-relaxation*. The optimal serial implementation of it executes in $O(n^k q^k)$ time (see [4]). We will now describe a variation of it in terms of the compatibility hypergraph.

Algorithm: k-relaxation
Input: An n-partite compatibility hypergraph $C = (V, E)$
Output: C_k

```
1. begin
2.    while $\exists$ a hyperclique $e$ :$(|e| = k - 1) \wedge$ (support-degree($e$) $< n - k + 1$)
3.    do
4.        $E' \leftarrow \{e' : e'$ is a hyperclique with $|e'| \leq k$ and $e \subseteq e'\}$
5.        $E \leftarrow E - E'$
6.    od
7. end
```

Theorem 6 *Let $\mathcal{N}$ be constraint network with n variables, each one having at most q values. Let also $k \leq n$ be a (constant) positive integer. Then k-relaxation can be executed serially in $O(n^k q^k)$ time and in parallel in $O(n^{k-1} q^{k-1} \log(nq))$ time on the CREW PRAM using $O(n^k q^k)$ processors.*

Proof: For $i \leq n$, there are $O(n^i q^i)$ hypercliques of size i in C. Therefore, C has $O(n^{k-1}q^{k-1})$ $(k-1)$-hypercliques, that can be identified in time $O(n^{k-1}q^{k-1})$. Let e be one of the $(k-1)$-hypercliques of C. Then, there are at most $q(n-k+1)$ hyperedges with k vertices that contain e. We can find the support-degree of e by checking each one of them. If the support-degree of e is $< n-k+1$, then there are at most $q(n-k)$ hypercliques that contain it, and are removed along with e. Thus, the serial complexity of k-relaxation is $O(n^k q^k)$. For the parallel complexity, each of the $O(n^{k-1}q^{k-1})$ $(k-1)$-hypercliques of C can be identified in constant time $O(k)$ using $O(2^{k-1})$ processors. For each such hyperclique e, we can check whether there are any hyperedges of size k in C that contain it in time $O(\log(nq))$ using $O(q(n-k+1))$ processors. If the support degree of e is less than $n-k+1$, then we can remove the corresponding k-hypercliques, that are at most $q(n-k)$, in time $O(1)$ using $2^{k-1}+q(n-k)$ processors. From the above, it follows that k-relaxation can be executed in $O(n^{k-1}q^{k-1}\log(nq))$ parallel time on the CREW PRAM using $O(n^k q^k)$ processors. ■

The processor bound stated in Theorem 6 is indeed high and the resulting algorithm is far from being cost optimal, since k-consistency can be achieved in $O(n^k q^k)$ optimal serial time. We believe that finding a cost optimal parallel algorithm that enforces k-concistency for any k is a non-trivial task and could constitute an interesting research problem on its own. Also, in all the theorems we state and prove in this section, there is some space for improvements on the processor bounds without affecting the asymptotic time complexity using Brent's scheduling technique, where a given set of processors shares the computational load previously allocated to a bigger number of processors, according to a suitably defined load sharing scheme.

Let us now extend the notion of partiality for any degree of consistency, and for constraint networks of any arity. A constraint network $\mathcal{N}$ is *α-partially k-consistent* if the following holds: for any set of values $\{d_{i_1}, \ldots, d_{i_{k-1}}\}$, such that every subset of it is compatible, there are at least $\alpha(n-k+1)$ domains not in $\{D_{i_1}, \ldots, D_{i_{k-1}}\}$ that contain at least one value d such that every subset of $\{d_{i_1}, \ldots, d_{i_{k-1}}\} \cup \{d\}$ is compatible, and each variable has at least one value compatible with other values in $\mathcal{N}$. The *α-partial k-consistency problem* is the problem of finding a maximal (with respect to the domains) α-partially k-consistent subnetwork $\mathcal{N}_{k,\alpha}$ of $\mathcal{N}$.

Let C be a compatibility hypergraph with n parts. A part of C whose vertices do not belong to any k-hyperclique is called a *k-empty* part. In terms of the compatibility hypergraph, the α-partial k-consistency problem is formulated as follows: Find the maximal (with respect to the set of vertices) induced subhypergraph $C_{k,\alpha}$ of C such that any $(k-1)$-hyperclique has support-degree $\geq \alpha(n-k+1)$ and no part of it is k-empty. Such a hypergraph is called *α-partially k-consistent*. Also, the *α-partial strong k-consistency problem* is the problem of finding a maximal (with respect to the set of vertices) induced subhypergraph $C^{\mathrm{s}}_{k,\alpha}$ of C that is α-partially i-consistent for any i, $i = 2, \ldots, k$, and any hyperedge e in $C^{\mathrm{s}}_{k,\alpha}$ of size at most k is a hyperclique. Notice that by restricting our attention to $C^{\mathrm{s}}_{k,\alpha}$, we do not lose any solution of the original CSP.

In order to solve the α-partial strong k-consistency problem, we introduce a modified version of the k-relaxation algorithm, called *strong (α, k)-relaxation*. Given a constant

$\alpha \in (0, 1]$ and a hypergraph C as input, the algorithm is the following:

```
Algorithm: strong $(\alpha, k)$-relaxation
Input:     An $n$-partite compatibility hypergraph $C$
Output:    $C^{s}_{k,\alpha}$
1.  begin
2.    while $\exists$ a hyperedge $e : (|e| \leq k) \wedge (e$ is not a hyperclique$)$
3.    do $E \leftarrow E - \{e\}$ od
4.    while $\exists$ a hyperclique $e :(|e| = k - 1)\wedge$
                               $(\text{support-degree}(e) < \alpha(n - k + 1))$
5.    do
6.      $E' \leftarrow \{e' : e'$ is a hyperclique with $|e'| \leq k$ and $e \subseteq e'\}$
7.      $E \leftarrow E - E'$
8.      if a $k$-empty part appears in $C$
9.      then terminate and return the empty hypergraph
10.   od
11.   for $i = k - 1$ downto 2
12.   do
13.     $E' \leftarrow\{e :(e$ is an $(i - 1)$-hyperclique$)\wedge$
                 $(\text{support-degree}(e) \leq \alpha(n - i + 1))\}$
14.     $E \leftarrow E - E'$
15.     if an $i$-empty part appears in $C$
16.     then terminate and return the empty hypergraph
17.   od
18. end
```

The first while-loop is a preprocessing stage that ensures that all hyperedges are hypercliques. This property guarantees that, during the execution of the rest of the algorithm, no hyperedge which is not a hyperclique appears. Also, the second while-loop is a variant of the k-consistency algorithm that outputs an α-partially k-consistent hypergraph. This hypergraph is subsequently transformed by the for-loop into a strongly consistent one. It is interesting to observe that in order to obtain strong consistency (i.e. i-consistency for $i < k$), the for-loop executes only the first parallel step of the variant of i-relaxation.

In fact, we prove the following:

Theorem 7 *Algorithm strong (α, k)-relaxation solves the problem of α-partial strong k-consistency and can be implemented to execute in $O(n^k q^k)$ serial time and in parallel in $O((nq)^{k-1} \log(nq))$ time using $O(n^k q^k)$ processors on the CREW PRAM.*

Proof: For the serial complexity of the algorithm, the observations of the proof of Theorem 6 apply here as well. However, we have also to take into account the additional cost of testing for the existence of i-empty parts, for $1 < i \leq k - 1$. This can be done as follows: During the identification of each of the $O(n^k q^k)$ hypercliques of C of size at most k, we increase k counters for each vertex v of C. The ith counter of v stores the number of the i-hypercliques that contain v. Now, whenever an i-hyperclique is removed, the value held in the i-th counter of each of its vertices is decreased by 1. An i-empty part appears if any counter reaches 0. This completes the proof of the serial complexity of the algorithm.

For the parallel complexity of the algorithm, and in a way similar to the proof of Theorem 6, we observe that there are $O(n^k q^k)$ hyperedges of size at most k. We can check whether any of these hyperedges is a hyperclique or not in $O(2^k)$ time. Also, it is obvious that the second while-loop is similar to the while-loop of the k-relaxation algorithm, and thus it has the same time and processor requirements. As for the identification of a k-empty part, it is easy to see that it can be accomplished in $O(\log(nq))$ time using $O(n^k q^k)$ processors.

In the same way, we can see that for each $i \leq k-1$, the steps that are executed during each iteration of the third while-loop, for each $i \leq k-1$, can be executed in $O(i \log(nq))$ time using $O(n^i q^i)$ processors on the CREW PRAM. This completes the proof of the time and processor complexity bounds of the algorithm.

To prove the correctness of the algorithm, let C_k denote the hypergraph that results after the execution of the two while-loops. Obviously, C_k is an α-partially k-consistent hypergraph. Therefore, it suffices to prove that after the execution of the for-loop on C_k, the resulting hypergraph is strongly k-consistent. We need the following lemma:

Lemma 2 *If the hypergraph C is α-partially r-consistent, for $r = i, \ldots, k$, then after the end of the $(k-i+1)$th execution of the for-loop, the resulting hypergraph C_{i-1} is an α-partially r-consistent hypergraph for $r = i-1, \ldots, k$.*

Proof: We denote by E_{i-2} the set of the hypercliques in C_i that have exactly $i-2$ vertices. Let $E_{i-2}^{incl} = \{e \in E_{i-2}\colon \exists e' \in E(C_i)\colon |e'| = i-1 \text{ and } e \subset e'\}$. That is, we define the set E_{i-2}^{incl} to be the set of these hypercliques e in E_{i-2} that are contained in some hyperclique with size equal to the size of e plus one. Finally, we define $E_{i-2}^{simpl} = E_{i-2} - E_{i-2}^{incl}$.

We first observe that for each $(i-1)$-hyperclique in C_i, and since it was not removed during any previous step, its support-degree in C_i is at least equal to $\alpha(n-i+1)$. Let now e be a hyperclique in E_{i-2}. If there exists an $(i-1)$-hyperclique e' in C_i such that e' contains the vertices of e, then, from the last observation, we get that in C_i:

$$\text{support-degree}(e) \geq 1 + \alpha(n-i+1) \geq \alpha(n-i+2) \tag{4}$$

Additionally, since no $(i-2)$-hyperclique $e \in E_{i-2}^{simpl}$ is contained in some bigger $(i-1)$-hyperclique, we get that in C_i:

$$\text{support-degree}(e) = 0 \tag{5}$$

Equations 4 and 5 imply that the $(k-i+1)$th iteration of the for-loop will remove exactly these hyperedges of C_i that belong in E_{i-2}^{simpl}. Now, since all the hyperedges with at least $i-1$ vertices are still hypercliques, all the r-hypercliques, where $r = i-2, \ldots, k-1$, of C_{i-1} have support-deegree $\geq \alpha(n-r)$. Thus, after the $(k-i+1)$th iteration of the for-loop, the resulting hypergraph C_{i-1} is also α-partially r-consistent for $r = i-1$. ■

Applying inductively Lemma 2 for the $k-2$ iterations of the for-loop, we conclude that the subgraph the algorithm returns is α-partially strongly k-consistent. ■

Before giving a faster algorithm for achieving strong partial local consistency, we present a generalization of Lemma 1.

Lemma 3 *Let C be the compatibility graph of a constraint network $\mathcal{N}$. Let q be the biggest cardinality of any part of C, Z be the number of hyperedges of C that have $k-1$ vertices, and $l_1, l_2 \leq n-k+1$ be two integers such that $l_2 < \frac{l_1}{1+q}$. If C contains at most m hypercliques with support-degree less than l_1, then C contains a hypergraph C' with at least $Z - \frac{m(l_1-l_2)}{l_1-l_2(1+q)}$ $(k-1)$-hypercliques each having support-degree at least l_2.*

Proof: The proof is similar to the proof of Lemma 1. We execute Steps 1–10 of strong (α, k)-relaxation, for $\alpha = (\frac{l_2}{n-k+1})$, on C, and let C' be the resulting hypergraph. We denote by R the set of $(k-1)$-hypercliques of C that are removed. Also, $R' \subseteq R$ denotes the set of hypercliques in R with support-degree $\geq l_1$ in C. Finally, we denote by E_R the set of k-hypercliques that contain two hypercliques e and e' such that $e' \in R'$, $e \in R$ and $e \neq e'$.

We first observe that the set R can be partitioned into two parts:

- The hypercliques that initially had support-degree $\geq l_1$, i.e. the hypercliques in R'.
- The hypercliques that initially had support-degree $< l_1$. From the hypothesis, these hypercliques are at most m.

It follows that:

$$|R| \leq m + |R'| \tag{6}$$

Let e be a hyperclique in R' that is removed at some step s. This hyperclique had initially support-degree at least l_1 in C. Observe that applying Steps 1–10 of strong (α, k)-relaxation for $\alpha = (\frac{l_2}{n-k+1})$ on C, removes hypercliques with support-degree less than l_2. Therefore, the number of k-hypercliques that contain the vertices in e and were removed at some step before the step s, is at least $l_1 - l_2$. Let e_k be one of these k-hypercliques. This hyperclique must have been removed from C during some step before step s. However, the removal of e_k was the result of the removal of *exactly* one $(k-1)$-hyperclique e' that is a subset of e_k, i.e. an $e' \in R$. Since e is removed at step s, e' cannot be e. But, then, each of the $l_1 - l_2$ hypercliques with k vertices that correspond to the k-hypercliques is contained in E_R.

Moreover, let e and e' be two $(k-1)$-hypercliques that are removed at different steps. We observe that the k-hypercliques that are removed as a result of removing e, are different from those that are removed as a result of removing e'. Therefore:

$$(l_1 - l_2)|R'| < |E_R| \tag{7}$$

Consider now a $(k-1)$-hyperclique e that is removed at some step s. Obviously, the number of k-hypercliques that contain the vertices of e as well as those of some other $(k-1)$-hyperclique $e' \neq e$ in R' that are removed *after* step s, is at most ql_2. Therefore, $|E_R| \leq ql_2|R|$, and, using Equation 6, we conclude that:

$$|E_R| \leq ql_2(m + |R'|) \tag{8}$$

From 7 and 8 we have the following:

$$\begin{aligned} (l_1 - l_2)|R'| &< ql_2(m + |R'|) \Rightarrow \\ |R'| &< \frac{ql_2 m}{l_1 - l_2(q+1)} \end{aligned}$$

Using 6, we get that

$$|R| < m + \frac{ql_2m}{l_1 - l_2(q+1)} \Rightarrow$$
$$|R| < m\frac{l_1 - l_2}{l_1 - l_2(q+1)}$$

therefore, C' has at least $Z - \frac{m(l_1-l_2)}{l_1-l_2(1+q)}$ $(k-1)$-hypercliques each having support-degree at least l_2. ■

We will now present a faster parallel algorithm that, given a compatibility hypergraph C with n parts each having at most q vertices, and a positive value $\alpha < 1/(q+1)$, outputs a subhypergraph of $C^{\mathrm{s}}_{k,\alpha}$ that contains C^{s}_k and is α-partially strongly k-consistent.

```
Algorithm: fast strong $(\alpha, k)$-relaxation
Input:     An $n$-partite compatibility hypergraph $C$
Output:    An $\alpha$-partially strongly $k$-consistent subhypergraph of $C^{\mathrm{s}}_{\alpha,k}$
           that contains $C^{\mathrm{s}}_k$, where $\alpha < \frac{1}{q+1}$
1.  begin
2.    $H \leftarrow \{e : e$ is a hyperclique of $C$ and $|e| = k-1\}$
3.    while $\exists H' \subseteq H : (|H'| > (nq)^{\frac{k-1}{2}}) \wedge$
                      $(\forall e \in H' :$ support-degree$(e) < n-k+1)$
4.    do
5.      in parallel remove from $C$ all hypercliques $e \in H'$
6.      $H \leftarrow H - H'$
7.      if a $k$-empty part appears
8.      then terminate and return the empty graph
9.    od
10.   run strong $(\alpha, k)$-relaxation
11. end
```

Theorem 8 *For any $\alpha < 1/(1+q)$, the above algorithm returns an α-partially strong k-consistent subhypergraph of $C^{\mathrm{s}}_{\alpha,k}$ which contains C^{s}_k. The algorithm runs in $O((nq)^{\frac{k-1}{2}}\log(nq))$ time using $O(n^kq^k)$ processors on the CREW PRAM.*

Proof: It is easy to see that the hypergraph the algorithm outputs is α-partially strongly k-consistent and contains C^{s}_k as a subgraph.

Also, each iteration of the for-loop can be executed in $O(\log(nq))$ time using $O(n^kq^k)$ processors on the CREW PRAM. During each such iteration, the algorithm removes at least $(nq)^{\frac{k-1}{2}}$ hypercliques of $k-1$ vertices. Therefore, at most $(nq)^{\frac{k-1}{2}}$ iterations of the for-loop are executed.

Now, from Lemma 3, since there are less than $(nq)^{\frac{k-1}{2}}$ hypercliques with support-degree $< n-k+1$, the strong (α, k)-relaxation will remove at most $(nq)^{\frac{k-1}{2}}\frac{1-\alpha}{1-\alpha-q\alpha}$ hyperedges with $k-1$ vertices. Thus, it will be executed in $O((nq)^{\frac{k-1}{2}}\log(nq))$ time using $O(n^kq^k)$ processors on the CREW PRAM, and the proof is completed. ■

6. Conclusion

Using as a departure point the fact that ACP is a P-complete problem, in conjunction with some implementations that indicate that solving ACP in parallel seems to require time linear in the number of variables using polynomially many processors, we examined some weaker versions of ACP and their complexity. We proposed several natural approximation schemes for ACP and showed them to be also P-complete. We managed to break the linearity barrier, by defining the concept of a partial solution, for both arc consistency and k-consistency, at the expense of allowing values that may be incompatible with every value in some fraction of the remaining variables. The fact that we managed to obtain sublinear parallel execution time only by defining a local consistency concept as weak as α-partial k-consistency, suggests that if we want to remove some non-trivial number of inconsistencies fast in parallel, we should concentrate on finding partial solutions, rather than approximate ones, even at the cost of allowing some inconsistencies.

However, there may be applications where partial solutions can be useful. For example, in overconstrained networks where a solution that satisfies all the given constraints may not exist, finding fast (in sublinear time) a partial solution may help to effect a good reduction on the search space. This further suggests an experimental line of research, where various classes of constraint networks can be subjected to algorithms enforcing α-partial k-consistency in order to test their effectiveness in eliminating sufficiently many inconsistencies.

The passage from approximation schemes to partial solutions is not a smooth one and it is an indication that in order to reach good parallel time bounds we have to lower somehow our demands, as far as the quality of the solution is concerned.

Erratum

An earlier version of the paper that appeared in [10] contained an erroneous proof to a theorem that appeared as Theorem 6 in that paper. Also, the same theorem was contained as Theorem 5 in [11].

Appendix

Proof of Theorem 2: We first present a logarithmic space reduction from the Planar Circuit Value Problem (PCVP) to the problem of deciding whether a specific value of a variable of a constraint network $\mathcal{N}$ belongs to the solution of the ACP, when the constraint graph G of $\mathcal{N}$ is planar with maximum degree 3. Briefly, PCVP consists in deciding whether a given boolean circuit, whose graph representation is planar, evaluates to 1 under a specific input value assignment. PCVP was proved to be P-complete [14]. We easily see that a gate of fan-out r may be replaced by a gate of fan-out 2, the outputs of which are directed to a binary tree of AND gates, of height $\leq \log r$. Thus, we can convert a circuit of gates of arbitrary fan-out to an equivalent circuit of gates of fan-out 2. Therefore, the Planar Circuit Value Problem with gates of fan-out 2 (PCVP2) is also P-complete. We will use PCVP2

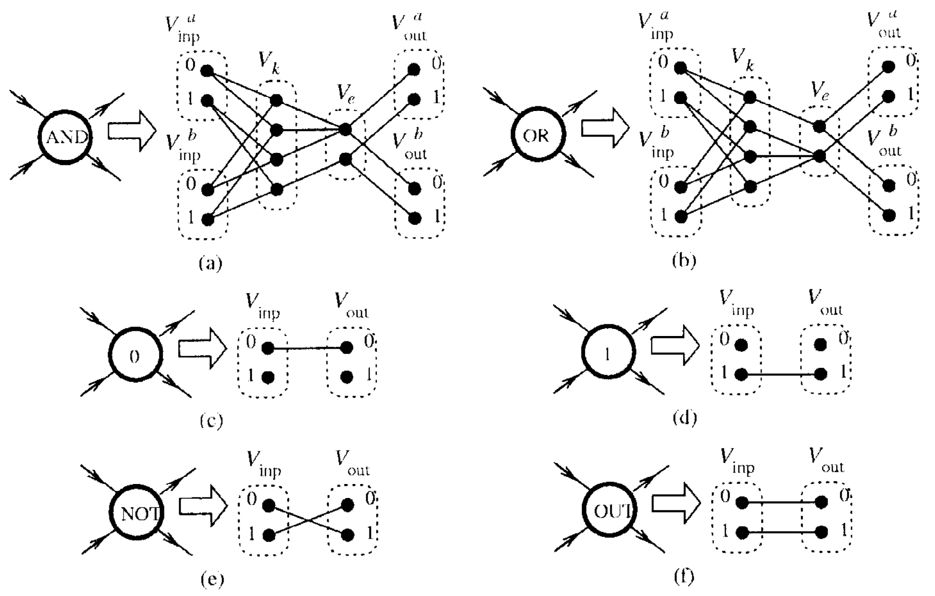

Figure 1. (a) The AND gadget, (b) the OR gadget, (c) the 0-input, (d) the 1-input, (e) the NOT gadget, and (f) the output gadget.

in order to present the reduction. For convenience, we will use the compatibility graph representation of constraint networks.

Let W be a circuit and $x_1, \ldots, x_k$ an input assignment to W. We will construct a compatibility graph C which has a vertex v with the following property: v belongs to the solution of the ACP if and only if the output of W is 1. Our reduction uses the gadgets shown in Figure 1. Each gadget that corresponds to an input gate, an output gate or a NOT gate, consists of two parts in C: V_{inp}, called the *input part* and V_{out}, called the *output part*. Also, each gadget that corresponds to an AND or an OR gate consists of 6 parts in C: V^a_{inp} and V^b_{inp} (called the *input parts*), V^a_{out} and V^b_{out} (called the *output parts*), and parts V_k and V_e. Each input or output part has two vertices: vertex 0 and vertex 1. The vertex for which we ask whether it belongs to the ACP solution or not, is the 1-vertex of the part V_{out} of the gadget that corresponds to the output gate of W.

For each edge (w, w') in W, the output part of the gadget that corresponds to gate w is the same as the input part of the gadget that corresponds to gate w'. Also, for any two parts V_1 and V_2 such that there is no edge from a vertex in V_1 to a vertex in V_2, we add to C all edges (v, v'), where $v \in V_1$ and $v' \in V_2$. We call such edges *full edges*, in constrast with the edges that are parts of the gadgets which we call *constraining edges*. Observe that if there are no constraining edges between two parts in the compatibility graph C, the constraint graph G has no edge between the vertices corresponding to the parts. Thus, by the construction of the gadgets, and since W is planar, G is also planar with maximum degree 3.

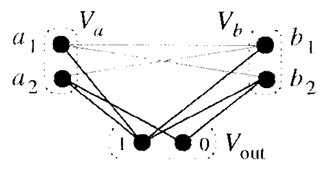

Figure 2. The gadget for deciding the emptiness of the ACP solution.

To prove the correctness of the reduction, observe that if the inputs of an AND gate are not *both* 1, then discrete relaxation will remove in the corresponding gadget the 1-vertices from parts V^a_{out} and V^b_{out} but not the 0-vertices. Similarly, if the inputs of an OR gate are not *both* 0, then discrete relaxation will remove in the corresponding gadget the 1-vertices of parts V^a_{out} and V^b_{out}. Finally, in a gadget that corresponds to a NOT gate, discrete relaxation will remove the 1-vertices of V_{out} iff the input to the gate is 1. It follows that the execution of discrete relaxation in C simulates the computation of the value of all the gates of the circuit W. Thus, in part V_{out} of the output of W, the 1-vertex will survive iff the value of W is 1.

Now in order to prove that it is also P-complete to decide whether the graph is emptied, we construct a new graph C', by adding to C two new parts V_a and V_b. Each one of them has two vertices, namely a_1, a_2 and b_1, b_2 respectively. We connect any vertex of V_a with any vertex of any other part but V_b and the part V_{out} of the output of W. Similarly, any vertex of V_b is connected to all other vertices except to the ones in parts V_a and V_{out}. Finally, we add the edges shown in Figure 2. It is now easy to verify that if discrete relaxation in graph C removes the 1-vertex from V_{out}, then it deletes all the vertices of graph C' and vice versa.

■

Proof of Theorem 3: If such a set D' could be found in NC for some $\epsilon < 1$, then $\epsilon|D'| \geq 1$ implies that $|D_{\text{AC}}| \geq 1$, i.e. discrete relaxation in $\mathcal{N}$ does not empty any domain. Also, if $\epsilon|D'| < 1$, then $|D'| < \lceil 1/\epsilon \rceil$, that is, D' contains a constant number $O(1/\epsilon)$ of values. Thus, using discrete relaxation on the restriction of $\mathcal{N}$ to D', we can conpute in constant time D_{AC}, and find in NC if discrete relaxation in $\mathcal{N}$ empties any domain. This, however, contradicts Theorem 2. ■

Proof of Theorem 4: Suppose that for some $\epsilon < 1$ there exists an NC algorithm A_{approx} that finds such a set R'. We apply algorithm A_{approx} to $\mathcal{N}$, and obtain a set $|R'|$. If $|R'| > 1$, then we remove in parallel the values in R', always checking if any domain is emptied, and apply again A_{approx}. We continue this loop until $|R'| \leq 1$. Since each application of A_{approx} removes at least a fraction ϵ of the values that discrete relaxation removes (i.e. of the values in R_{AC}), and $|R_{\text{AC}}| \leq nq$, the loop iterates at most $\log(nq)$ times. Upon exiting the loop, since $|R'| \leq 1$, the network contains at most $1/\epsilon$ values not in $\mathcal{N}_{\text{AC}}$. These can be removed, by applying discrete relaxation in parallel, in constant time. In this way, we can compute $\mathcal{N}_{\text{AC}}$ in NC, a contradiction unless P=NC. ■

Proof of Theorem 5: For a positive integer $r \leq n-1$, we call *r-elimination* the $(\frac{r}{n-1})$-discrete relaxation on the compatibility graph.

We will work with an equivalent problem in the compatibility graph C of $\mathcal{N}$. Thus, for a given vertex v of C, we define $\text{elimin}(v, C)$ to be the least h such that a $(n-h)$-elimination does not remove v and does not empty any part of C. Obviously, if value d corresponds to vertex v, then $\text{elimin}(d, \mathcal{N}) = \text{elimin}(v, C)$. In this way, we are interested in finding a value $\text{elimin}(v, C)_{\text{approx}}$ such that:

$$\lambda \cdot \text{elimin}(v, C) \geq \text{elimin}(v, C)_{\text{approx}} \geq \text{elimin}(v, C)$$

Our proof uses a reduction from the problem of the *Propositional Horn Satisfiability* (PHSP). The problem is defined for sets of *propositional Horn clauses*, where a propositional Horn clause is either:

- a propositional atom of the form Q, which is called an *assertion*,
- a formula of the form $P \leftarrow P_1, \ldots, P_i$, which is called an *implication*,
- or a propositional negative atom of the form $\neg P$, which is called a *goal*.

An atom P appearing in a set $\mathcal{S}$ of propositional Horn clauses is *solvable* iff P appears in an assertion, or there is an implication $P \leftarrow P_1, \ldots, P_i$ and all of $P_1, \ldots, P_i$ are solvable in at least one of the implications or assertions of $\mathcal{S}$. In PHSP, we are given a set of propositional Horn clauses which contain only a single goal (such a set of clauses is called a *propositional logic program*), and we are asked to determine whether the atom that appears in the goal is solvable (in which case the program is *unsatisfiable*) or not.

PHSP is P-complete, even if we restrict ourselves to logic programs with implications that have at most two atoms in the right handside of the formula. For a more detailed description of the problem, see [17].

We now present a construction that, given arbitrary integers $m > l$, for any given instance $\mathcal{P}$ of the PHSP with at most two atoms in the right handside of any implication, produces a compatibility graph C with a vertex f' that corresponds to the goal of the PHSP and has the following properties:

$$\text{The goal of } \mathcal{P} \text{ is solvable} \Rightarrow \text{elimin}(f', C) = m + 1 \tag{9}$$

$$\text{The goal of } \mathcal{P} \text{ is not solvable} \Rightarrow \text{elimin}(f', C) = l + 1 \tag{10}$$

Our construction uses a collection of gadgets which are shown in Figure 3. The basic idea of the construction is the following: for each atom R that appears in a formula of $\mathcal{P}$, we construct m parts of vertices. Each such part contains two vertices, t and f. The intuition behind the two-vertex parts is that if, during some elimination procedure in C, vertex f is removed from *all* the parts corresponding to R, then this implies that R is solvable. Moreover, we choose vertex f' to be any vertex f of an arbitrary part of the m parts that are constructed for the goal Q of $\mathcal{P}$.

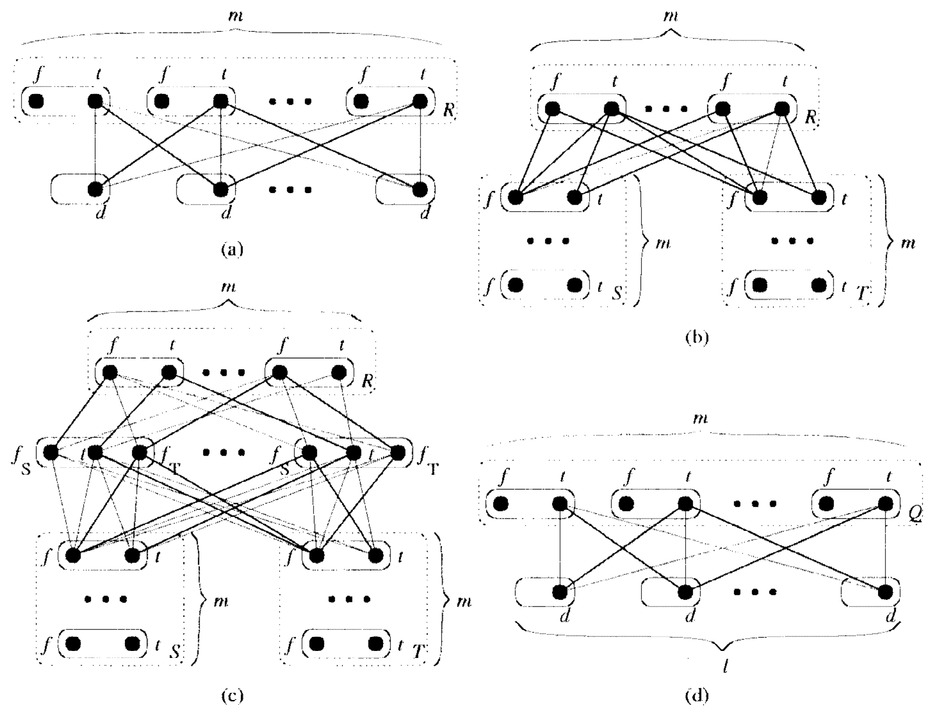

Figure 3. The gadgets (a) for the assertion R (b) for implications $R \leftarrow S$ and $R \leftarrow T$, with only one atom on the right handside, (c) for implication $R \leftarrow S, T$, with two atoms on the right handside, and (d) for the goal Q.

In the sequel, and for notational convenience, we will say that a vertex v of an n-partite graph has *lack of support* equal to an integer i when there are exactly i parts of the graph with no vertex connected to v.

For any assertion R, our construction (see Figure 3.a) introduces m additional dummy parts. These parts have only one vertex d, and are connected which the t vertices in the m parts of R, but not to the f vertices. Thus, for any atom that is solvable by an assertion, the f vertices of the parts that correspond to it have in C lack of support equal to m.

For any implication of the form $R \leftarrow S$, our construction (see Figure 3.b) ensures that the f vertices of the parts corresponding to R are connected only with the f vertices of the parts corresponding to S. Again, the t vertices of R are connected to all vertices of S. Thus, if the f vertices are removed from the parts corresponding to S, the f vertices of R obtain lack of support equal to m.

For any implication of the form $R \leftarrow S, T$, our construction (see Figure 3.a) introduces an intermediate level of m parts with three vertices, namely t, f_S and f_T. Whenever the f vertices of the parts corresponding to both S and T are removed, then in each three-vertex part both f_S and f_T have lack of support equal to m; if f_S and f_T are removed from the graph, then the f vertices of the parts corresponding to R are also left with lack of support equal to m.

We also add a set a set of l dummy parts with only one vertex d each. Each vertex d is connected to the t vertices of the parts corresponding to the goal Q of $\mathcal{P}$ (see Figure 3.b). In this way, the f vertices of the parts corresponding to Q have in C lack of support equal to l. Therefore, $\text{elimin}(f', C) \geq l + 1$.

If for two parts V_1 and V_2 of C the gadgets do not indicate any connection among their vertices, we connect V_1 and V_2 with the complete bipartite graph. Thus, the t vertices, the d vertices corresponding to the assertions, and the d vertices corresponding to the dummy parts, are connected pairwise. Hence, no part remains empty when performing any elimination of the vertices of C.

Let n denote the total number of parts in C. It is clear that the $(n - 1 - m)$-elimination will not remove from C any f vertices of the parts corresponding to solvable atoms, but the $(n-m)$-elimination will remove all such f vertices. Thus, if the goal Q is solvable, then f' is not removed by the $(n-1-m)$-elimination, but it is removed by the $(n-m)$-elimination, so in this case $\text{elimin}(f', C) = m + 1$. Suppose now, that Q is not solvable. The f vertices of the parts corresponding to Q have in C, because of the dummy parts, lack of support equal to $l < m$. In this way a $(n - 1 - l)$-elimination will not remove the f vertices of the parts corresponding to Q, but a $(n - l)$-elimination will remove them, so in this case $\text{elimin}(f', C) = l + 1$. This proves the properties of Equations 9 and 10.

Suppose now, towards a contradiction, that we can approximate $\text{elimin}(v, C)$ in NC by a factor $\lambda > 1$. Then, we construct the graph C for integers m and l such that $\frac{m+1}{l+1} > \lambda$, and compute $\text{elimin}(f', C)_{\text{approx}}$. If $\text{elimin}(f', C)_{\text{approx}} \geq m + 1$, and since $\frac{m+1}{l+1} > \lambda$, we get that $\frac{m+1}{l+1} \cdot \text{elimin}(f', C) > \text{elimin}(f', C)_{\text{approx}}$. Thus, if $\text{elimin}(f', C) = l + 1$, then $m + 1 > \text{elimin}(f', C)_{\text{approx}}$, a contradiction. Therefore, $\text{elimin}(f', C) = m + 1$ and, by Equation 9, the PHSP goal is solvable.

If, on the other hand, $\text{elimin}(f', C)_{\text{approx}} < m+1$ then $\text{elimin}(f', C)$ cannot be $m+1$, since we would also have that $\text{elimin}(f', C)_{\text{approx}} \geq m+1$, a contradiction. Thus, $\text{elimin}(f', C) = l + 1$ and, by Equation 10, the PHSP goal is unsolvable.

In this way, it is possible to decide in NC the satisfiablity of PHSP, which is a contradiction unless P=NC. ■

Acknowledgments

We acknowledge the helpful comments of the anonymous referees, that were crucial in improving the presentation of our results.

References

1. Anderson, R. and Mayr, E. (1987). Parallelism and greedy algorithms. *Advances in Computing Recearch* 4: 17–38. See also: A P-complete problem and approximations to it, Technical Report, Dept. Computer Science, Stanford University, California (1984).
2. Bessiere, C. (1994). Arc-consistency and arc-consistency again. *Artificial Intelligence* 65: 179–190.

3. Cohen, D. A., Cooper, M. C., and Jeavons, P. G. (1994). Characterizing tractable constraints. *Artificial Intelligence* 65: 347–361.

4. Cooper, M. C. (1990). An optimal k-consistency algorithm. *Artificial Intelligence* 41: 89–95.

5. Cooper, P. R., and Swain, M. J. (1992). Arc consistency: parallelism and domain dependence. *Artificial Intelligence* 58: 207–235.

6. Dechter, R. (1992). Constraint networks. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, 2nd ed., pages 276–285, Wiley, New York.

7. Dechter, R. and Meiri, I. (1994). Experimental results of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence* 68: 211–241.

8. Dechter, R. and Pearl, J. Tree clustering for constraint networks. *Artificial Intelligence* 38: 353–366.

9. Dendris, N. D., Kalafatis, I. A., and Kirousis, L. M. (1994). An efficient parallel algortihm for geometrically characterising drawings of a class of 3-D objects. *Journal of Mathematical Imaging and Vision* 4: 375–387.

10. Dendris, N. D., Kirousis, L. M., Stamatiou, Y. C, and Thilikos, D. M. Partiality and approximation schemes for local consistency in networks of constraints. In P. S. Thiagarajan, editor, *Proc. of the 15th Conference on the Foundations of Software Technology and Theoretical Computer Science (FST & TCS) (Bangalore, 1995)*, Vol. 1026 of Lecture Notes in Computer Science, pages 210–224, Springer-Verlag.

11. Dendris, N. D., Kirousis, L. M., Stamatiou, Y. C, and Thilikos, D. M. Partial arc consistency. In *Over-Constrained Systems (Cassis, 1995)*, Vol. 1106 of Lecture Notes in Computer Science, pages 229–236, Springer-Verlag.

12. Freuder, E. C. (1990). Complexity of k-tree structured constraint satisfaction problems. *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 4–9, Boston, Mass.

13. Gibbons, A. and Rytter, W. (1988). *Efficient Parallel Algorithms*. Cambridge University Press.

14. Goldschlager, L. M. (1977). The monotone and planar circuit value problems are log-space complete for P. *SIGACT News* 9(2): 25–29.

15. Van Hentenryck, P., Deville, Y., and Teng, C. M. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* 57: 291–321.

16. Karp, R. M. and Ramachandran, V. (1990). Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam.

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

18. Kasif, S. and Delcher, A. L. Local consistency in parallel constraint satisfaction networks. *Artificial Intelligence* 69: 307–327.

19. Kirousis, L. M. Fast parallel constraint satisfaction. *Artificial Intelligence* 64: 147–160.

20. Lieberherr, K. J. and Specker, E. (1981). Complexity of partial satisfaction. *J. of the ACM* 28: 411–421.

21. Mackworth, A. K. (1992). Constraint satisfaction. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, 2nd ed., pages 285–293, Wiley, New York.

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

23. Mackworth, A. K. and Freuder, E. C. (1993). The complexity of constraint satisfaction revisited. *Artificial Intelligence* 59: 57–62.

24. Samal, A. and Henderson, T. C. (1987). Parallel consistent labeling algorithms. *International Journal of Parallel Programming* 16(5): 341–364.

25. Waltz, D. (1975). Understanding line drawings of scenes with shadows. *Psychology of Computer Vision*, pages 19–91, McGraw-Hill, New York.

26. Zhang, Y. and Mackworth, A. K. (1991). Parallel and distributed algorithms for finite constraint satisfaction problems. *Proceedings 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 394–397, Dallas, TX.