On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks

Simon Kasif
Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218, USA

ABSTRACT

Constraint satisfaction networks have been shown to be a very useful tool for knowledge representation in Artificial Intelligence applications. These networks often utilize local constraint propagation techniques to achieve local consistency (consistent labeling in vision). Such methods have been used extensively in the context of image understanding and interpretation, as well as planning, natural language analysis and truth maintenance systems. In this paper we study the parallel complexity of discrete relaxation, one of the most commonly used constraint propagation techniques. Since the constraint propagation procedures such as discrete relaxation appear to operate locally, it has been previously believed that the relaxation approach for achieving local consistency has a natural parallel solution. Our analysis suggests that a parallel solution is unlikely to improve the known sequential solutions by much. Specifically, we prove that the problem solved by discrete relaxation (arc consistency) is log-space complete for P (the class of polynomial-time deterministic sequential algorithms). Intuitively, this implies that discrete relaxation is inherently sequential and it is unlikely that we can solve the polynomial-time version of the consistent labeling problem in logarithmic time by using only a polynomial number of processors. Some practical implications of our result are discussed. We also provide a two-way transformation between AND/OR graphs, propositional Horn satisfiability and local consistency in constraint networks that allows us to develop optimal linear-time algorithms for local consistency in constraint networks.

1. Introduction

Constraint satisfaction networks have been shown to be a very useful tool for knowledge representation in Artificial Intelligence applications [19]. These networks often utilize local constraint propagation techniques to achieve global consistency. Such methods have been used extensively in the context of image understanding and interpretation [7, 12, 15] as well as planning, natural language analysis and commonsense reasoning [19].

In particular, this paradigm has been applied to solve the *constraint satisfaction problem* (CSP) which is a key problem in many Artificial Intelligence

Artificial Intelligence **45** (1990) 275–286
0004-3702/90/$03.50

applications. The constraint satisfaction problem can be informally defined as follows. Let S be a set of objects. Each object has a set of possible labels associated with it. Additionally, we are given a set of constraints that for each object s and label x describe the compatibility of assigning the label x to object s with assignment of any other label x' to any other object s'. Intuitively, we have to produce an assignment of labels to objects which is consistent with all the constraints.

Since the CSP is known to be NP-complete, the discrete relaxation method has been proposed to reduce the initial ambiguity and achieve local consistency. Local consistency or arc consistency (see formal definition in the next section) allows us to assign a label x to an object s iff for any other object s' in the domain there exists a valid assignment of a label x' which does not violate the constraints (a formal definition is given in the next section). This formalization allows us to achieve local consistency by local propagation of constraints. Specifically, a discrete relaxation algorithm can discard a label from an object if it is incompatible with all other possible assignments of labels to the remaining objects. The discrete relaxation approach has been successfully applied to numerous computer vision applications [2, 4, 11, 18]. The sequential time complexity of arc consistency is discussed in [13].

In this paper we study the parallel complexity of arc consistency. Since the constraint propagation procedures such as discrete relaxation appear to operate locally, it has been previously believed that the relaxation approach for the CSP has a natural parallel solution [1, 15, 19]. Our analysis suggests that a parallel solution is unlikely to improve the known sequential solutions by much. Specifically, we prove that the problem of achieving local consistency belongs to the class of inherently sequential problems called log-space complete for P (P-complete).

Intuitively, a problem is log-space complete for P iff a logarithmic-time parallel solution for the problem will produce a logarithmic-time parallel solution for any polynomial-time deterministic sequential algorithm. This implies that, unless $P \subseteq NC$ (the class of problems solvable in logarithmic parallel time with polynomial number of processors), we cannot solve the problem in logarithmic time using a polynomial number of processors. This result is based on the "parallel computation thesis" proved in [6] that establishes that parallel time computation is polynomially related to sequential space. Specifically, the class of problems that can be solved in logarithmic parallel time with polynomial number of processors is equivalent to the class of problems that can be solved in polynomial time using logarithmic space on a sequential machine. For length considerations, we assume that the reader is familiar with elementary complexity theory and log-space reducibility techniques [5] and the literature on discrete relaxation (network consistency algorithms). For completeness we shall provide the necessary definitions in the next two sections.

2. Constraint Satisfaction Networks and Discrete Relaxation

The constraint satisfaction problem and arc consistency are formally defined in [11]. For completeness we give a semiformal definition here. Let $V = \{v_1, \ldots, v_n\}$ be a set of variables. With each variable v_i we associate a set of labels L_i. Now let P_{ij} be a binary predicate that defines the compatibility of assigning labels to objects. Specifically,

$$P_{ij}(x, y) = 1$$

iff the assignment of label x to v_i is compatible with the assignment of label y to v_j. The constraint satisfaction problem (CSP) is defined as the problem of finding an assignment of labels to the variables that does not violate the constraints given by P_{ij} (consistent with all the constraints). More formally, a solution to the CSP is a vector $(x_1, \ldots, x_n)$ such that x_i is in L_i and, for each i and j, $P_{ij}(x_i, x_j) = 1$.

The standard approach to model CSPs is by means of a constraint graph. The nodes of the constraint graph correspond to the variables of the CSP. The edges of the graph correspond to the binary constraints in the CSP. That is, with each edge in the constraint graph we associate a matrix that shows which assignments of labels to variables on the arc are permitted. Thus, the size of the input is $O(EK^2)$, where E is the number of edges in the graph and K is the number of labels.

For example, the four-queens problem can be seen as an instance of a CSP. To confirm this, associate a variable with each column in the board and let $L_i = \{1, 2, 3, 4\}$ for $1 \leq i \leq 4$. Let $P_{ij}(x, y) = 1$ iff positioning of a queen in row x at column i is "safe" when there is a queen in column j and row y.

As mentioned in the introduction, the CSP is known to be NP-complete. Therefore several polynomial approximation algorithms were proposed and were shown to perform quite well in practical applications. The most significant class of algorithms are variations on discrete relaxation [15], also known as network consistency algorithms [12]. Formally, a locally consistent solution to the CSP (arc consistency, abbreviated as AC in the remainder of the paper) is a set of sets $M_1, \ldots, M_n$ such that M_i is a subset of L_i and a label x is in M_i iff for every M_j, $i \neq j$, there is a y_{xj} in M_j, such that $P_{ij}(x, y_{xj}) = 1$. Intuitively, a label x is assigned to a variable iff for every other variable there is at least one valid assignment of a label to that other variable that supports the assignment of label x to the first variable. We call a set $M_1, \ldots, M_n$ a maximal solution for an AC problem iff there does not exist any other solution $S_1, \ldots, S_n$ such that $M_i \subseteq S_i$ for all $1 \leq i \leq n$. We are only interested in maximal solutions for AC. This restriction is necessary since any AC problem has a trivial solution: the set of empty sets. Additionally, recall that any solution for an AC problem represents a set of candidate solutions for the original CSP, which will eventually be verified by a final exhaustive check. Thus, by insisting on

maximality we guarantee that we are not losing any possible solutions for the original CSP. Therefore, in the remainder of this paper a solution for an AC problem is identified with a maximal solution.

The standard approach for achieving arc consistency is by repeatedly applying the following procedure (discrete relaxation):

PAR-AC

for each arc from X_i to X_j of the constraint graph test whether for each label l for X_j there exists l' for X_i that permits it.

It is easy to see that this algorithm will terminate in $O(EK^2nK)$ or equivalently $O(nEK^3)$ time, where E is the number of edges in the graph and K is the number of labels (recall that EK^2 is the size of the input). In fact, in [13] and this paper much better sequential algorithms for the problem are developed.

3. The Complexity of Searching AND/OR Graphs

In this section we state several preliminary definitions and results that will be used in the following section to analyze the complexity of AC. We begin by defining AND/OR graphs [14].

Definition 3.1. An *AND/OR graph* is a six-tuple (A, O, E, s, S, F) where A is a set of AND-nodes, O is a set of OR-nodes, E is a set of directed edges connecting nodes in $A \cup O \cup S \cup F$, s is a unique start node in A, S is a set of success nodes and F is a set of failure nodes. The solvability of a node in an AND/OR graph is defined recursively:

- If x is an S-node, it is solved.
- If x is an AND-node, then it is solved iff all its successors (defined by the direction of the edges of E) are solved.
- If x is an OR-node then it is solved iff one of its successors is solved.

An AND/OR graph has a solution iff s is solved.

Proposition 3.2 (Jones and Laaser [8]). *Finding a solution for an AND/OR graph is log-space complete.*

Proof. This result can be obtained by observing that GAME studied in [8] is an instance of the problem of AND/OR solvability. □

We now define the class of propositional Horn clauses.

Definition 3.3. A propositional formula H is said to be a *propositional Horn clause* iff one of the following holds:

- H is a propositional atom of the form Q, called an assertion.

- H is a propositional formula of the form $P \leftarrow P_1, \& \cdots \& P_n$, denoted by $P \leftarrow P_1, \ldots, P_n$ and called an implication.
- H is a propositional negative atom (literal) of the form $\neg P$, denoted by $\leftarrow P$ and called a goal.

We note that our slightly restrictive definition of Horn clauses (not allowing multiple literals in the goal) does not restrict the expressiveness of the language.

Definition 3.4. A *propositional logic program* is a set of propositional Horn clauses with a single goal. We define the unsatisfiability of a set of propositional Horn clauses S as a solvability relation on the set of propositional names in S. The definition is recursive:

- If P is an assertion in S, then it is solvable.
- If P appears on the left-hand side in a set of implications of the form

$$P \leftarrow P_1, \ldots, P_n,$$

then P is solvable iff each one of the P_i is solvable in at least one of the implications.

A propositional logic program is *unsatisfiable* iff the propositional name that appears in the single negative atom is solvable. The problem of testing whether a propositional logic program is unsatisfiable will be referred to as the propositional Horn satisfiability problem (PHSP).

Example 3.5. The following set of Horn clauses is unsatisfiable since P is solvable:

$\leftarrow P.$
$P \leftarrow Q,R.$
$P \leftarrow S,T.$
$R \leftarrow S.$
$T \leftarrow P.$
$Q.$
$S.$

The next theorem, though not explicitly stated previously in a published form, is part of the common folklore among theoreticians [17].

Theorem 3.6 (folklore). *The problem of testing the satisfiability of propositional Horn clauses is log-space complete.*

Proof. The proof is by reduction from solvability of AND/OR graphs (GAME of

Jones and Laaser [8]) and will not be presented here in full detail. Generally, "reduction" is the most common technique to show a problem X is log-space complete. Specifically, it is adequate to show the problem is in P (the class of polynomial-time algorithms), and then reduce a known log-space complete problem to X using a function computable in logarithmic space (log-space) by a deterministic Turing machine. Since log-space reducibility is a transitive relation, we can then deduce that if we had a logarithmic-time parallel algorithm to solve X, we could also perform every other sequential polynomial-time computation in logarithmic time. In our case the reduction of AND/OR graph solvability to propositional Horn satisfiability is immediate (in some sense it is the same problem). We label all the nodes in the AND/OR graph with distinct propositional atoms. Then for each AND-node P connected to $P_1, \ldots, P_n$ we create a formula $P \leftarrow P_1, \ldots, P_n$. For each OR-node P connected to $P_1, \ldots, P_n$ we create formulae

$$\begin{aligned} P &\leftarrow P_1 . \\ &\vdots \\ P &\leftarrow P_n . \end{aligned}$$

For each terminal success node Q we create the assertion Q. Finally if the start node of the graph is labeled by P we add the goal $\leftarrow P$ to the set.

It is easy to see that the original graph has a solution iff the set of formulae created in this fashion is unsatisfiable and the transformation can indeed be done in log-space. □

It is not difficult to verify that the following is also true:

Theorem 3.7. *Theorem* 3.6 *holds for propositional logic programs restricted to implications that have at most two atoms on the right-hand side of the implication.*

4. The Complexity of AC

In this section we present a two-way reduction between PHSP and AC. This will establish our main result, namely that the problem of finding a solution to AC is log-space complete (AC is P-complete). Additionally, this will show that AC can be solved by a linear-time algorithm developed previously for the PHSP.

To show AC is P-complete we show that AC is in P and subsequently prove that satisfiability of propositional Horn clauses (PHSP) is reducible to AC. The first part of the proof is straightforward since PAR-AC as presented above has polynomial-time sequential complexity (see [13]). In fact the edge consistency algorithm as given in [12] is linear in the number of edges in the constraint graph [13].

Theorem 4.1. *The propositional Horn clause satisfiability problem (PHSP) is log-space reducible to AC.*

Proof. Let Pr be a propositional logic program such that no implication has more than two atoms on its right-hand side. We will also assume that all the atoms in Pr are uniquely labeled with integer values. We shall construct an AC problem G from Pr such that Pr is unsatisfiable iff a unique variable $\langle P_0 \rangle$ in G that corresponds to the unique goal $\leftarrow P_0$ does not have a valid assignment of the label f. The AC problem is constructed in the following way.

(1) For each atom A in Pr, we create a unique variable $\langle A \rangle$.
(2) For each assertion Q in Pr we create a unique variable $\langle \text{SOLVED}_Q \rangle$.
(3) Create a unique variable $\langle P_0 \rangle$ that corresponds to the goal $\leftarrow P_0$ of Pr.
(4) For each implication of the form $P \leftarrow Q,R$. We add the variable $\langle Q, R \rangle$ to G.

This construction defines all the variables of G. The initial label sets are created as follows:

- Each variable, with the exception of the $\langle \text{SOLVED} \rangle$ variables, is assigned the label l.
- For each assertion Q in Pr we add the label q to the initial label set of $\langle \text{SOLVED}_Q \rangle$.
- For each variable of the form $\langle R \rangle$ we add the label f to its initial set.
- For each variable of the form $\langle S, T \rangle$ we add the labels f_S and f_T to its initial set.

We are now ready to define constraints of the problem G. We define the constraints using a compatibility matrix COM, whose entries are of the form

$$\text{COM}[\text{variable}, \text{variable}, \text{label}, \text{label}] .$$

$\text{COM}[v_i, v_j, x, y] = 1$ iff the assignment of label y to variable v_j is compatible with the assignment of label x to variable v_i. An alternative natural representation is to use a directed multigraph where the nodes correspond to the variables of the problem and the edges are labeled with the constraints of the problem. It is important to observe that in order to preserve log-space reducibility we do not need to create the entire compatibility matrix COM. For a full description of the AC problem we only need to create a list of the constraints of the form $\text{COM}[\text{var}, \text{var}, \text{label}, \text{label}] = 0$. That is, we describe only the incompatible assignments. The remaining entries in the matrix can be filled with 1s.

For each implication of the form $P \leftarrow R$ we add the constraints:

$$\text{COM}[\langle P \rangle, \langle R \rangle, f, f] = 1 ,$$

$$\text{COM}[\langle P \rangle, \langle R \rangle, f, l] = 0 .$$

For each implication of the form $P \leftarrow Q,R$ we add the constraints:

$$\begin{aligned}
&\text{COM}[\langle P\rangle, \langle Q,R\rangle, f, f_Q] = 1\,,\\
&\text{COM}[\langle P\rangle, \langle Q,R\rangle, f, f_R] = 1\,,\\
&\text{COM}[\langle P\rangle, \langle Q,R\rangle, f, l] = 0\,,\\
&\text{COM}[\langle Q,R\rangle, \langle R\rangle, f_R, f] = 1\,,\\
&\text{COM}[\langle Q,R\rangle, \langle R\rangle, f_R, l] = 0\,,\\
&\text{COM}[\langle Q,R\rangle, \langle Q\rangle, f_Q, f] = 1\,,\\
&\text{COM}[\langle Q,R\rangle, \langle Q\rangle, f_Q, l] = 0\,.
\end{aligned}$$

Finally, for every variable of the form $\langle \text{SOLVED}_Q\rangle$ we add the constraint

$$\text{COM}[\langle Q\rangle, \langle \text{SOLVED}_Q\rangle, f, q] = 0\,.$$

This completes the definition of all the "necessary" constraints of the AC problem. The rest of the matrix COM can be filled with 1s.

Note that the label f must be removed from all the variables that correspond to the assertions of the logic program. Using induction on the length of the satisfiability proof it is fairly easy to show that the label f will be removed from the variable $\langle P_0\rangle$ that corresponds to the goal $\leftarrow P_0$ iff P_0 is solvable. The formal proof is omitted.

Now we have to verify that the above construction can be done using only logarithmic space on the work tape of the Turing machine. We shall sketch the main ideas of the proof method. If the construction were to be carried out in the order given above it would have taken linear space (linear in the number of total occurrences of all the atoms in Pr). Fortunately, since we assumed the

$$\begin{aligned}
&\text{COM}[\langle P\rangle, \langle Q,R\rangle, f, l] = 0\\
&\text{COM}[\langle P\rangle, \langle S,T\rangle, f, l] = 0\\
&\text{COM}[\langle Q,R\rangle, \langle R\rangle, f_R, l] = 0\\
&\text{COM}[\langle Q,R\rangle, \langle Q\rangle, f_Q, l] = 0\\
&\text{COM}[\langle S,T\rangle, \langle S\rangle, f_S, l] = 0\\
&\text{COM}[\langle S,T\rangle, \langle T\rangle, f_T, l] = 0\\
&\text{COM}[\langle R\rangle, \langle S\rangle, f, l] = 0\\
&\text{COM}[\langle T\rangle, \langle P\rangle, f, l] = 0\\
&\text{COM}[\langle Q\rangle, \langle \text{SOLVED}_Q\rangle, f, q] = 0\\
&\text{COM}[\langle S\rangle, \langle \text{SOLVED}_S\rangle, f, s] = 0
\end{aligned}$$

Fig. 1. Constraints for Example 4.2.

atoms were initially numbered by integers, we can follow the above construction in a demand-driven fashion as explained below.

To start off we can create all the variables of the form $\langle Q \rangle$ and their respective label sets. This can be done with logarithmic-space consumption since processing each one of the N-variables we need $(\lg N)$-bits. For each assertion we can add the respective label to $\langle \text{SOLVED} \rangle$. For implications of the form $P \leftarrow Q,R$ we generate a new variable and its respective initial label set. This step requires a counter that can be implemented in logarithmic space. Finally, for each implication encountered we can generate the constraints (again in logarithmic space). This completes the generation of all the necessary (see above discussion) information that completely describes the AC problem G. □

Example 4.2. Consider the following PHSP:

$\leftarrow P.$
$P \leftarrow Q,R$
$P \leftarrow S,T.$
$R \leftarrow S.$
$T \leftarrow P.$
$Q.$
$S.$

We construct the following AC problem. The variables of the problem are: $\langle P \rangle$, $\langle Q \rangle$, $\langle R \rangle$, $\langle S \rangle$, $\langle T \rangle$, $\langle Q,R \rangle$, $\langle S,T \rangle$, $\langle \text{SOLVED}_Q \rangle$, $\langle \text{SOLVED}_S \rangle$. The initial assignments of labels are as follows:

$\langle P \rangle$: $\{f, l\}$
$\langle Q \rangle$: $\{f, l\}$
$\langle R \rangle$: $\{f, l\}$
$\langle S \rangle$: $\{f, l\}$
$\langle T \rangle$: $\{f, l\}$
$\langle Q,R \rangle$: $\{f_Q, f_R, l\}$
$\langle S,T \rangle$: $\{f_S, f_T, l\}$
$\langle \text{SOLVED}_Q \rangle$: $\{q\}$
$\langle \text{SOLVED}_S \rangle$: $\{s\}$

Finally, the constraints are given in Fig. 1.

4.1. Reducing AC to PHSP achieves an optimal time algorithm for AC

In this section we shall give a linear-time reduction from AC to PHSP. It is also possible to show that this reduction is log-space, but we could not find any AI

applications for this particular theoretical result. Thus, for simplicity of presentation we provide the linear-time reduction only. Theorems 4.1 and 4.3 establish that the two problems have essentially the same complexity (sequential and parallel) and allow us to derive a simple linear-time solution to AC.

Theorem 4.3. *Arc consistency (AC) is linear-time reducible to the propositional Horn clause satisfiability problem (PHSP).*

Proof. The reduction is similar to the one used in Theorem 4.1. Instead of presenting a formal proof, we sketch the construction for a simple constraint network that consists of three variables X, Y and Z. We observe that a label l drops from X iff all the labels that are compatible with l at Y drop or all the labels compatible with l at Z drop. We express this statement in propositional Horn logic. We let $P_{X,l}$ be a proposition that is true iff l drops from X. Let $P_{Y,l_1,l_2,\ldots}$ be a proposition that states that $l_1, l_2, \ldots$ drop from Y, where $l_1, l_2, \ldots$ is the set of labels compatible with l at X. Let $P_{Z,l_1,l_2,\ldots}$ be a proposition that states that $l_1, l_2, \ldots$ drop from Z, where $l_1, l_2, \ldots$ is the set of labels compatible with l at X. It is clear that we can express the conditional statement that l drops from X in propositional Horn form by the following formulae:

$$P_{X,l} \leftarrow P_{Y,l_1,l_2,\ldots}.$$

$$P_{X,l} \leftarrow P_{Z,l_1,l_2,\ldots}.$$

$$P_{Y,l_1,l_2,\ldots} \leftarrow P_{Y,l_1}, P_{Y,l_2}, \ldots.$$

$$P_{Z,l_1,l_2,\ldots} \leftarrow P_{Z,l_1}, P_{Z,l_2}, \ldots.$$

Now, we apply one iteration of algorithm PAR-AC to the network to construct the set (list) of all labels that will be dropped in the first iteration. For all such labels we construct a proposition of the form $P_{V,l}$. E.g, if l_1 is dropped from variable Y in the first iteration of PAR-AC we construct a proposition (assertion) of the form P_{Y,l_1}. We claim, that a label l is dropped from a variable V iff the proposition $P_{V,l}$ is provable from the program we constructed. The proof can be obtained by a simple induction on the number of iterations of PAR-AC.

It is easy to see that this reduction can be accomplished in linear time. One iteration of PAR-AC requires $O(EK^2)$ time, where E is the number of edges in the graph and K is the number of labels. Thus, constructing all the assertions of the program will require $O(EK^2)$ time. The construction of the rules of the propositional program requires one traversal of all the edges of the constraint graph. Each "1" in the compatibility matrix associated with an edge corresponds to exactly one predicate symbol occurrence on the right-hand side of an

implication in the propositional program we create. Thus, the entire construction can be done in linear time. □

Theorem 4.4. *ARC consistency (AC) can be solved in $O(EK^2)$ sequential time.*

Proof. By Theorem 4.3 we can convert any AC problem to a PHSP in linear time such that the resulting program is of size $O(EK^2)$. It is known that PHSP can be solved by a simple linear-time algorithm (essentially by a modified topological sort algorithm). □

Theorem 4.4 is a minor improvement of the result published in by Mackworth and Freuder [13] who report an $O(EK^3)$-time algorithm. A similar complexity result was also independently derived by Samal and Anderson [16]. However, here we derive the result by a simple reduction to PHSP.

It is worth noting that Theorem 4.3 establishes a precise connection between the basic iteration step of discrete relaxation (PAR-AC) and the fixpoint operator associated with every logic program. Thus, algorithm PAR-AC can be seen a special case of the bottom-up algorithm that computes the fixpoint of a propositional logic program. We observe the connection between constraint networks and fixpoint operators in discrete lattices in [10]. Similar connections have been pointed out by Bible [3] where it has been shown that constraint networks can be seen as a special case of function-free logic programs (datalog programs). Our construction is stronger since it shows a reduction to the simplest possible class of logic programs, namely, propositional logic programs. Additionally, the connection is tight in the complexity sense, namely, the reduction from AC to PHSP and back can be accomplished in linear time.

5. Conclusion

In this paper we showed a tight connection between the problem of achieving local consistency in propositional constraint networks and the problem of satisfiability of propositional Horn clauses. The immediate corollary of our reductions is that a very important class of algorithms (discrete relaxation) which were previously believed to be highly parallelizable are in fact inherently sequential. This negative worst-case result needs to be quantified. Essentially, it suggests that the application of massive parallelism will not change significantly the worst-case complexity of discrete relaxation (unless one has an exponential number of processors). However, this result does not preclude research in the direction of applying parallelism in a more controlled fashion. For instance, we can easily obtain speedups when the constraint graph is very dense (the number of edges is large). With $O(E)$ processors we can always achieve $O(nK)$, performance by a brute-force parallelization of PAR-AC (see details in [9]). Speedups are also possible in the case where the number of

processors is significantly smaller than the size of the constraint graph (a very likely case). In this case, it may be possible to obtain a full P-processor speedup. We are currently actively investigating this interesting case.

ACKNOWLEDGEMENT

Thanks are due to Johan de Kleer, Alan Mackworth and Azriel Rosenfeld for their constructive comments that contributed to the final form of this paper. The initial stages of the research reported in the paper were supported by NSF under grant DCR-18408 while the author was a visiting scientist at the Center for Automation Research, University of Maryland. This research was also supported by the Air Force Office of Scientific Research under grant AFOSR-89-1151, National Science Foundation under grant IRI-88-09324 and NSF/DARPA under grant CCR-8908092.

REFERENCES

1. D.H. Ballard and C.M. Brown, *Computer Vision* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
2. H.G. Barrow and J.M. Tenenbaum, MSYS: A system for reasoning about scenes, Tech. Note 121, SRI AI Center, Menlo Park, CA (1976).
3. W. Bible, Constraint satisfaction from a deductive viewpoint, *Artificial Intelligence* **35** (1988) 401–413.
4. R.A. Brooks, Symbolic reasoning among 3-D models and 2-D images, *Artificial Intelligence* **17** (1981) 285–348.
5. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to NP-Completeness* (Freeman, San Francisco, CA, 1979).
6. L.M. Goldschlager, A unified approach to models of synchronous parallel machines, in: *Proceedings 10th Symposium on Theory of Computing* (1978) 89–94.
7. R.M. Haralick and L.G. Shapiro, The consistent labeling problem: Part I, *IEEE Trans. Pattern Anal. Mach. Intell.* **1** (1979) 173–184.
8. N. Jones and T. Laaser, Complete problems for deterministic polynomial time, *Theor. Comput. Sci.* **3** (1977) 105–117.
9. S. Kasif, Parallel solution for constraint satisfaction problems, in: *Proceedings Conference on Principles of Knowledge Representation* (1989).
10. S. Kasif and A. Rosenfeld, The fixpoints of images and scenes, in: *Proceedings 1983 Conference on Computer Vision and Pattern Recognition* (1983).
11. L.J. Kitchen, Relaxation applied to matching quantitative relational structures, *IEEE Trans. Syst. Man Cybern.* **10** (1980) 96–101.
12. A.K. Mackworth, Consistency in networks of relations, *Artificial Intelligence* **8** (1977) 99–118.
13. A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction, *Artificial Intelligence* **25** (1985) 65–74.
14. N.J. Nilsson, *Problem-Solving Methods in Artificial Intelligence* (McGraw-Hill, New York, 1971).
15. A. Rosenfeld, R. Hummel, and S. Zucker, Scene labeling by relaxation operations, *IEEE Trans. Syst. Man Cybern.* **6** (1976) 420–433.
16. A. Samal and T. Henderson, Parallel consistent labelling algorithms, *Int. J. Parallel Program.* **16** (1987) 341–364.
17. J. Ullman, Personal communication (1985).
18. D. Waltz, Understanding line drawings of scenes with shadows, in: P.H. Winston, ed., *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975) 19–92.
19. P.H. Winston, *Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984).

Received October 1985; revised version received March 1990