Reduction Operations for Constraint Satisfaction*

ROBERT M. HARALICK,†
LARRY S. DAVIS,††
AZRIEL ROSENFELD,

and

DAVID L. MILGRAM

Computer Science Center, University of Maryland, College Park, Maryland 20742

Communicated by A. Rosenfeld

ABSTRACT

The problem of labeling a set of units in such a way as to satisfy an order-N compatibility relation is discussed and shown to be NP-complete. Some general procedures for reducing the size of the compatibility relation are defined, properties of these procedures are derived, and their incorporation into a backtracking process is discussed. Their computational cost and storage requirements are also considered.

0. INTRODUCTION

The problem of classifying or labeling a set of objects in such a way that given constraints are satisfied arises in a variety of applications. The constraints can be expressed in the form of relations that specify allowable (or "consistent") combinations of (object, label) pairs. Finding consistent labelings is an NP-complete problem; it can be done using a straightforward backtracking approach, but this is computationally expensive.

This paper formulates the problem of finding consistent labelings as defined by an order-N relation on the (object, label) pairs, and defines some general procedures for reducing the size of this relation. Some properties of these procedures are derived, and their incorporation into a backtracking process is discussed.

*The support of the National Science Foundation under Grant MCS-76-23763 and AFOSR Grant 77-3307 is gratefully acknowledged, as is the help of Mrs. Shelly Rowe in preparing this paper.

†Permanent address: University of Kansas, Lawrence, KS 66045.
††Present address: University of Texas, Austin, TX 78712.

© Elsevier North-Holland, Inc., 1978 0020-0255/78/0014-0199/$01.75

1. THE LABELING PROBLEM

Consider a finite set U of *units*, and another finite set L of *labels*. A *labeling* of U is a mapping from U into L. We determine a set of allowable labelings by specifying an order-N *compatibility relation* $R \subseteq (U \times L)^N$. Specifically, let $(u_1,\ldots,u_K)$ be a K-tuple of elements of U, and let $(l_1,\ldots,l_K)$ be the corresponding labels in L; then we say that this labeling of $u_1,\ldots,u_K$ is *consistent* if, for every N-tuple $u_{i_1},\ldots,u_{i_N}$ of these u's, the $2N$-tuple

$$u_{i_1}, l_{i_1}, u_{i_2}, l_{i_2}, \ldots, u_{i_N}, l_{i_N}$$

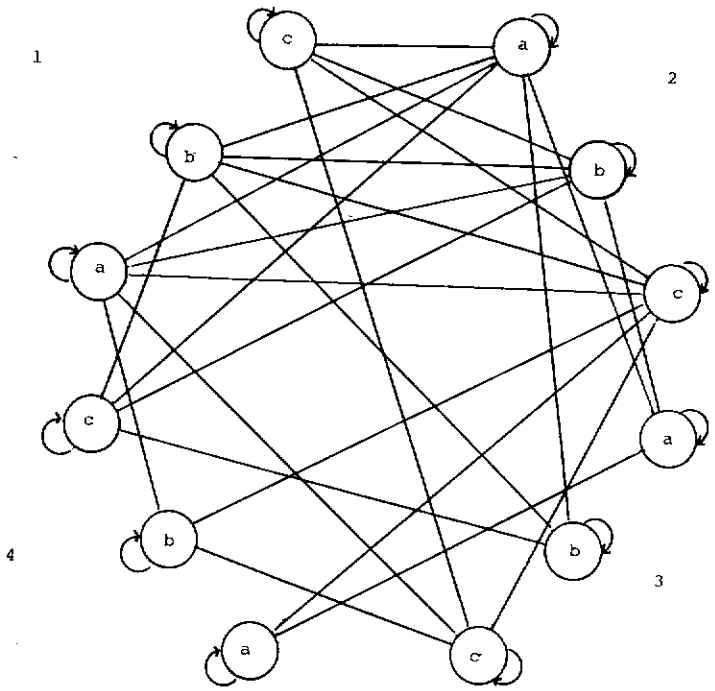
is in R. If $U=\{u_1,\ldots,u_M\}$, then a consistent labeling of $(u_1,\ldots,u_M)$ is called a *globally consistent* labeling (GCL) of U. The constrained *labeling problem* is, given U, L, and R, to find the (possibly empty) set of GCLs.

Figure 1 gives a simple example of the labeling problem. The unit set $U=\{1,2,3,4\}$, the label set $L=\{a,b,c\}$, and the compatibility relation is of order 2. A compatibility relation of order 2 can be represented as a graph. Figure 1 shows such a graph along with a table which defines R. Figure 2 displays the graph of Figure 1 so that it is easier to discover the two globally consistent labelings of $\{1,2,3,4\}$: (a,c,c,b) and (b,a,b,c). Note that they correspond to the cliques of size 4.

A careful examination of Fig. 1 or 2 indicates that no labeling of unit 1 constrains any labeling of unit 2. Since the links which constrain the possible labels of unit 1 and 2 are superfluous, we simplify the graph by drawing only those links which constrain the labelings of their respective units. Figure 3 shows this alternative representation of the compatibility constraint of Figure 1. Thus, if a link is not drawn between two units, then all possible labelings of one unit are compatible with all possible labelings of the second unit. Conversely, if a link is drawn between units i and j, then the pair of compatible labels for units i and j corresponds to the relation P_{ij} where $P_{ij}=\{(l_1,l_2)\in L\times L|(i,l_1,j,l_2)\in R\}$.

A variety of techniques can be used to solve the labeling problem. Among these are methods involving backtracking, as well as the brute-force technique of "generate and test". In the next few paragraphs we give a brief description of the backtracking approach and illustrate how it can be used to solve the labeling problem for Fig. 1.

Backtracking accompoishes a depth-first search of a forest of consistent labelings of the units. (See Fig. 4.) In the following discussion of the backtracking procedure, refer to Fig. 5. In its first step, label a for unit 1 is instantiated by the procedure. Since $(1,a,1,a)\in R$, the choice gives a compatible labeling of unit 1. In its next step, label a for unit 2 is instantiated. Since $(2,a,2,a)$, $(1,a,2,a)$, and $(2,a,1,a)\in R$, the labeling (a,a) for units $(1,2)$ is consistent and the link from $1a$ to $2a$ in the tree is allowed to remain. In its third step, label a



R	
1a	1a, 2a, 2b, 2c, 3c, 4b
1b	1b, 2a, 2b, 2c, 3b, 4c
1c	1c, 2a, 2b, 2c, 3c
2a	1a, 1b, 1c, 2a, 3a, 3b, 4c
2b	1a, 1b, 1c, 2b, 3a, 4c
2c	1a, 1b, 1c, 2c, 3c, 4a, 4b
3a	2a, 2b, 3a, 4a
3b	1b, 2a, 3b, 4c
3c	1a, 1c, 2c, 3c, 4b
4a	2c, 3a, 4a
4b	1a, 2c, 3c, 4b
4c	1b, 2a, 2b, 3b, 4c

Fig. 1. The graph and table representation of the second-order compatibility relation $R \subseteq (\{1,2,3,4\} \times \{a,b,c\})^2$.

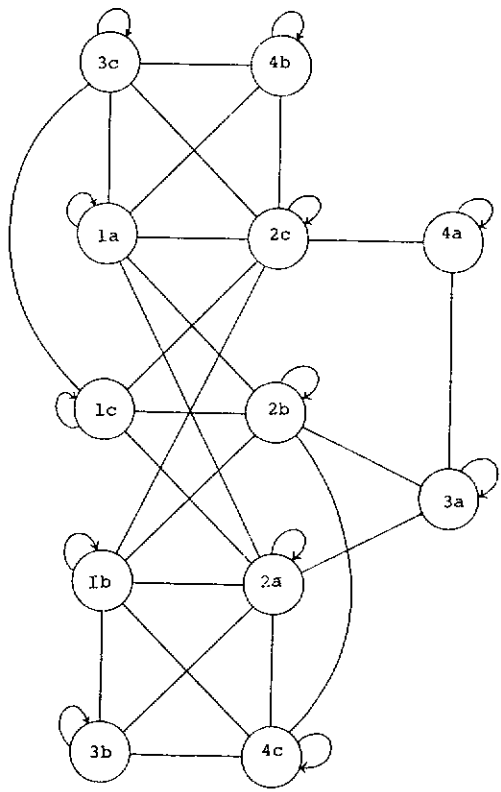
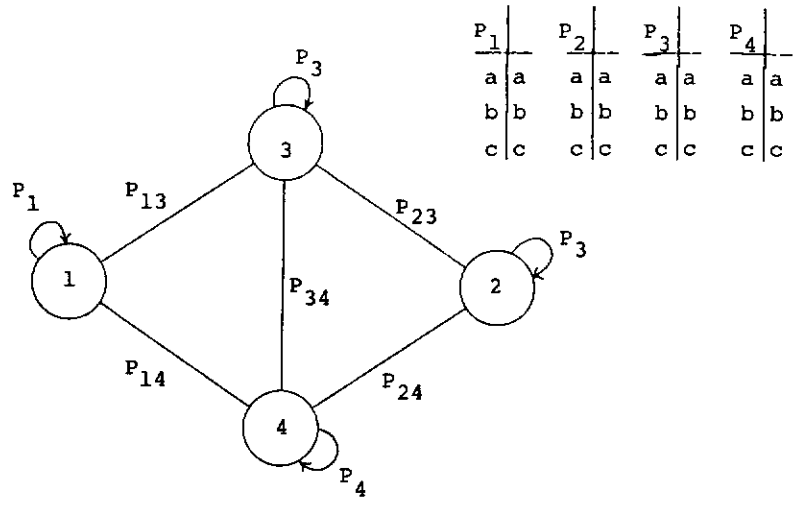


Fig. 2. Reorganization of the graph of Fig. 1 which shows clearly the two globally consistent labelings of $\{1,2,3,4\}$. They correspond to the two cliques of size 4 in the graph. They are (a,c,c,b) and (b,a,b,c) for units $(1,2,3,4)$.

for unit 3 is instantiated. But since $(1,a,3,a)$ is not in R, all paths in the tree including $3a$ and below can be omitted from the search. Next label b for unit 3 is instantiated. But since $(1,a,3,b)$ is not in R, all paths in the tree including $3b$ and below can be omitted from the search. Likewise for label c of unit 3.

Now all labels for unit 3 have been instantiated and all have failed. Therefore, the backtracking returns to the most recently visited unit that still has uninstantiated labels and instantiates the next possible lable for it. Thus, label b for unit 2 is instantiated next. It is consistent. Then label a for unit 3 is instantiated. It is consistent. Then label a for unit 4 is instantiated. It is not consistent, since $(1,a,4,a)$ is not in R. Likewise labels b and c for unit 4 are not consistent. So label b for unit 3 is instantiated. It is not consistent. Then label c for unit 3 is instantiated. It is also not consistent.



P_{13}	
a	c
b	b
c	c

P_{14}	
a	b
b	c
c	-

P_{23}	
a	a,b
b	a
c	c

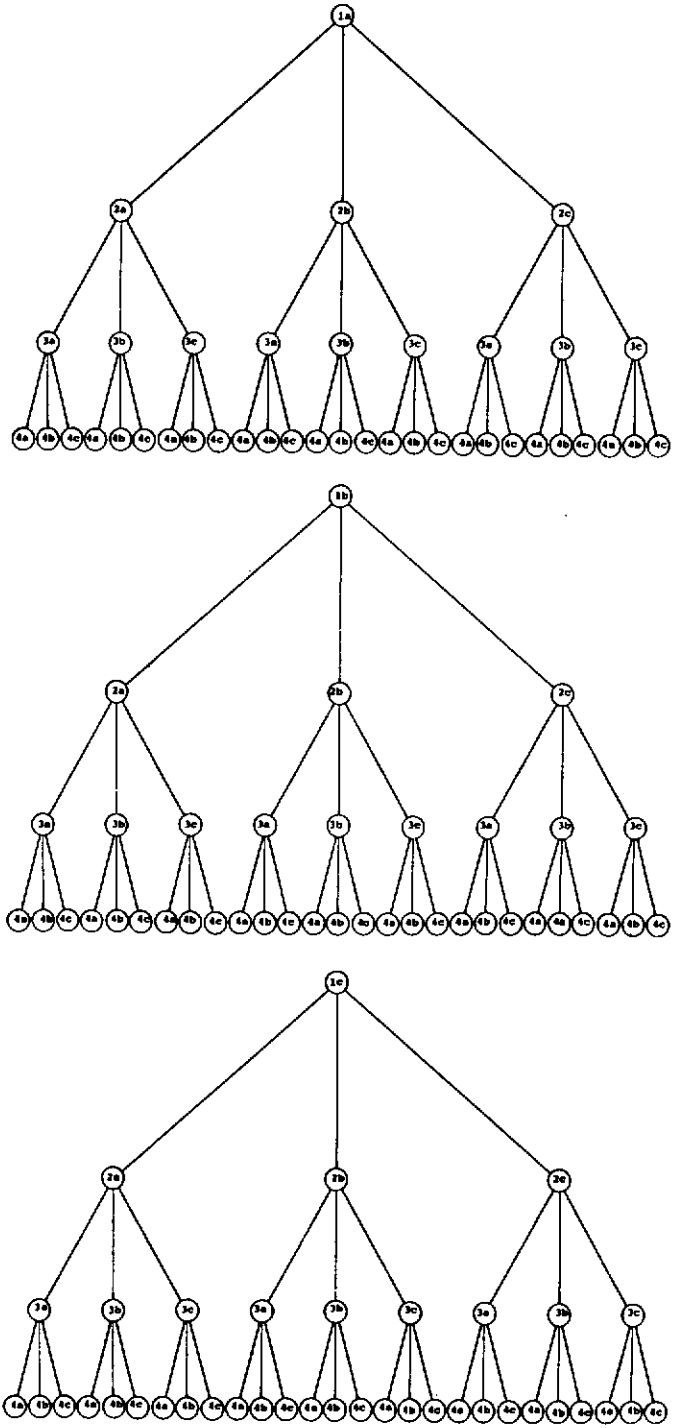
P_{24}	
a	c
b	c
c	a,b

P_{34}	
a	a
b	c
c	b

Fig. 3. An alternative representation of the compatibility constraint of Fig. 1. Links are drawn only between units whose labels can constrain one another. The label (i,j) gives the name of binary relation constraint $P_{ij} = \{(l_1, l_2) \in L \times L | (i, l_1, j, l_2) \in R\}$.

The failure of all labels of unit 3 causes the next label, label c, of unit 2 to be instantiated. Labels a and b for unit 3 are not consistent. But label c for unit 3 is consistent. Then label b for unit 4 is the only consistent label for it. At this point the backtracking has determined that the labeling (a,c,c,b) is the only globally consistent labeling of units $(1,2,3,4)$ if unit 1 is constrained to have label a. For the remainder of the backtracking, labels b and c for unit 1 are instantiated and the search proceeds in a similar manner.

The efficiency of backtracking is due to the fact that as soon as a label for a unit is found to be inconsistent, all labelings below that one in the tree do not have to be searched, since they too will be inconsistent. However, backtracking suffers from thrashing. Examing Figure 5, we see that label b for unit 3 is instantiated three times and fails three times simply for the reason that labeling (a,b) for units $(1,2)$ is not compatible. Mackworth [1] calls compatibility constraints which allow this to happen "arc inconsistency constraints".



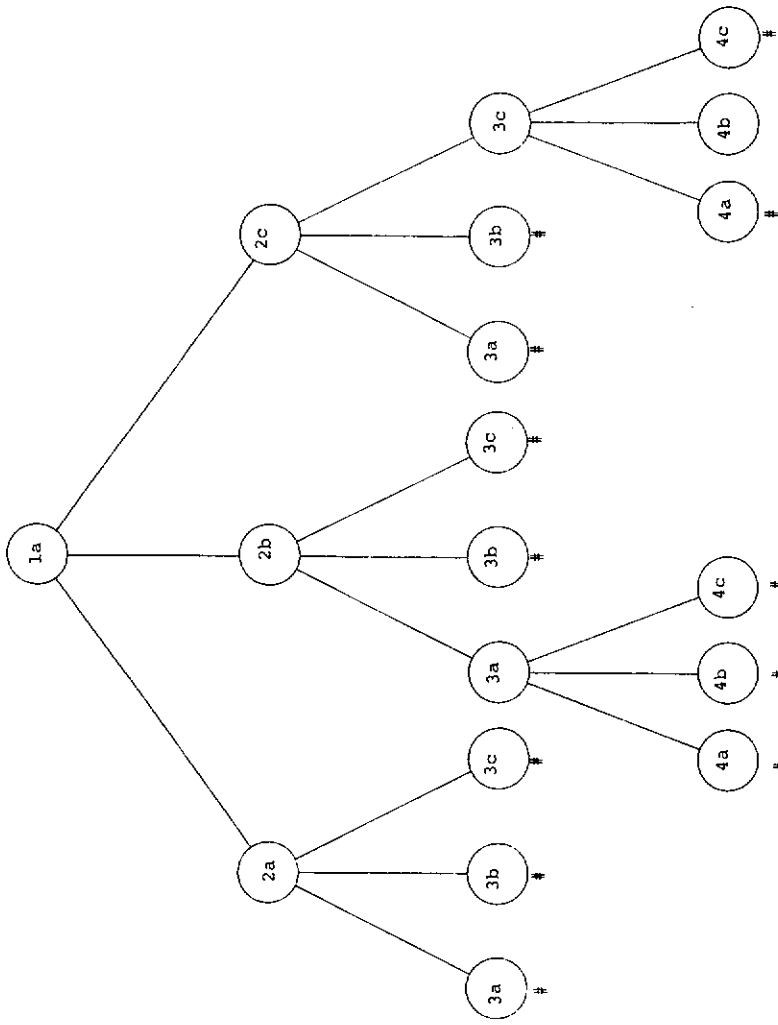


Fig. 5. The extent to which the tree of Fig. 4 has to be searched in order to discover the globally consistent labelings for Fig. 1 when unit 1 is constrained to have label *a*. The contradiction sign # appears below the first node on a path at which a labeling inconsistency occurs.

Thrashing can be reduced by eliminating elements of the compatibility relation which are "false trails". These are elements which do not contribute to any globally consistent labeling. The winnowing out of elements from R can be accomplished by iterative procedures of various kinds as well as by some one-pass algorithms. In this paper we shall derive general procedures which reduce R while preserving all globally consistent labelings, thus reducing the thrashing behavior of the backtracking procedure.

Unfortunately, we cannot expect these procedures to significantly reduce the worst-case computational cost of solving the labeling problem, because the labeling problem is NP-complete, as will be shown immediately below. The most we can hope for is that such procedures will eliminate some of the more easily located "false trails" in the original compatibility constraint R.

THEOREM 1. *The labeling problem is NP-complete.*

Proof. First we show that the labeling problem is solvable in polynomial time by a non-deterministic Turing machine. This is immediate, since the Turing machine can non-deterministically choose a labeling and then verify its global consistency by checking that every N-tuple of units has labels that are allowed by the compatibility constraint. Since N, the factor size of the relation R, is fixed, the number of N-tuples to be checked is $\binom{M}{N} \leq M^N$, which is polynomial in M.

Now we show that some NP-complete problem would be solved by solving the labeling problem. The problem to be considered is the k-colorability problem, whose NP completeness is proved in [2]. Briefly, a graph G is k-colorable if there exists an assignment of the integers $1,\ldots,k$ to the vertices of G such that no two adjacent vertices are assigned the same integer. Any colorability problem can be encoded as a labeling problem as follows: Let $G=(V,E)$ be the graph to be colored. A labeling problem is defined by a unit set U, a label set L, and an N-ary constraint relation $R \subseteq (U \times L)^N$. Let $U=V$, $L=\{1,\ldots,k\}$, and $R \subseteq (U \times L)^2$ be defined by

$$R = \left\{ (u_1, l_1, u_2, l_2) \in (U \times L)^2 \middle| (u_1, u_2) \in E \text{ implies } l_1 \neq l_2 \right\}.$$

Thus if $(l_1,\ldots,l_m)$ is a GCL of U, then every labeled pair (u_i,l_i,u_j,l_j) is in R, i.e., if $(u_i,u_j) \in E$, then $l_i \neq l_j$. Thus no two adjacent nodes in G are given the same color. ■

2. PREVIOUS WORK

Much of the fundamental mathematical analysis of constraint satisfaction was given by Montanari [3]. He showed that the reduction of R to a smallest subrelation S having the same set of GCLs as R was, in general, NP-complete. He therefore suggested that approximations to S that can be computed in

polynomial time should be investigated. In particular, he introduced the notion of the largest *path-consistent* subrelation (P) of a binary relation R as follows:

Let $R \subseteq (U \times L)^2$, and let $r = (u_i, l_i, u_j, l_j) \in R$. Let $\rho = (u_i = u_1, u_2, \ldots, u_p, \ldots, u_n = u_j)$ by any sequence of units (i.e., a path). Then r is *allowed by* ρ if there are labels $(l_i = l_1, l_2, \ldots, l_n = l_j)$ such that $(u_k, l_k, u_{k+1}, l_{k+1}) \in R$ for $1 \leqslant k < n$. We say r is *legal* if it is allowed by all paths containing u_i and u_j, and finally that R is *path-consistent* if every $r \in T$ is legal.

If we fix n, then we can let P_n denote the largest path-consistent subrelation of R for paths of length n. Montanari showed that $P_n = P_{n'}$ for all $n, n' \geqslant 3$. The proof involves a straightforward induction on n. Thus, we will simply refer to P as the path-consistent subrelation of R. Section 3 contains generalizations of the notion of path consistency.

It is obvious that $S \subseteq P$, but the containment is ordinarily proper. A special case of path consistency called *arc consistency* is defined by considering only r's of the form (u_i, l_i, u_i, l_i) and restricting $n = 3$. We will let P_a denote the largest arc-consistent subrelation of a relation R. Intuitively, this corresponds to checking all unit-label assignments to ensure that for every other unit, a compatible label exists at that unit. Waltz's [4] programs, which apply constraints to line drawings of blocks-world scenes with shadows, compute the arc-consistent subrelation of the original binary relation defined over the vertices of a blocks-world scene.

Mackworth [1] presented efficient iterative algorithms to compute P and P_a. Section 4 describes generalizations to n-ary relations of Mackworth's algorithm. Rosenfeld et al. [5, 6] showed that P_a can be computed by a parallel, iterative procedure ("discrete relaxation") and introduced both fuzzy and probabilistic generalizations ("probabilistic relaxation") of the constraint analysis problem.

An important application of the procedures to compute P and P_a is their incorporation into search procedures to compute the set of GCLs of a relation R. Mackworth [1], Gaschnig [7], and Haralick [8] discuss this for discrete constraints, while Barrow and Tenenbaum [9] and Davis [10] treat fuzzy constraints. Section 4 contains a discussion of search in the case of discrete n-ary relations.

Many of the results presented in this paper are elucidations and generalizations of work described by Mackworth [1] and Haralick [8], among others. The generalization from binary to n-ary constraints is straightforward, but it does provide a unified notation in which a variety of constraint-satisfaction problems can be formulated.

3. REDUCTION OF THE COMPATIBILITY RELATION

In this section we investigate an approach to solving the labeling problem based on reducing the size of the compatibility relation R, which in turn

reduces the computational cost of procedures such as backtracking. We first show that there is a smallest order-N relation $S_R \subseteq (U \times L)^N$ that defines the same set of GCL's that R does, and we then describe some methods of eliminating $2N$ -tuples form R that are not in S_R. Let us denote by $\mathfrak{L}(R)$ the set of GCL's defined by R.

PROPOSITION 1. *Let* $R' \subseteq R \subseteq (U \times L)^N$; *then* $\mathfrak{L}(R') \subseteq \mathfrak{L}(R)$.

PROPOSITION 2. *Let* $R_1, R_2 \subseteq (U \times L)^N$ *be such that* $\mathfrak{L}(R_1) = \mathfrak{L}(R_2)$; *then* $\mathfrak{L}(R_1 \cap R_2) = \mathfrak{L}(R_1)$.

Since U and L are finite, it follows from Proposition 2 by induction that if we take the intersection of all relations $R_i \subseteq (U \times L)^N$ that have a given GCL set $\mathfrak{L}(R)$, then this intersection also has the same GCL set. Clearly this intersection is the smallest relation that has GCL set $\mathfrak{L}(R)$; we denote it by S_R, and call it the *reduced relation* corresponding to R. In summary, we have

COROLLARY 3. *Let* $S_R = \cap_{\mathfrak{L}(R_i) = \mathfrak{L}(R)} R_i$; *then* $\mathfrak{L}(S_R) = \mathfrak{L}(R)$.

Evidently $S_R \subseteq R$, since R is one of the R_i's. As an immediate consequence of Proposition 1 and Corollary 3 we have

COROLLARY 4. *Let* $R' \subseteq (U \times L)^N$ *satisfy* $S_R \subseteq R' \subseteq R$; *then* $\mathfrak{L}(R') = \mathfrak{L}(R)$.

Let $(l_1, \ldots, l_M)$ be a GCL of $U = \{u_1, \ldots, u_M\}$; then for every N-tuple $u_{i_1}, \ldots, u_{i_N}$ of the u's, the $2N$-tuple

$$(u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N})$$

must be in any $R' \subseteq (U \times L)^N$ that has the given set of GCLs, and in particular, every such $2N$-tuple must be in S_R. Conversely, the set of all such $2N$-tuples, for all GCLs of U, constitutes a relation $\subseteq (U \times L)^N$. Clearly this relation has the given set of GCLs, and is the smallest relation having this set of GCLs; hence this relation must be S_R. We have thus proved

PROPOSITION 5. $S_R = \{(u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N}) | (l_1, \ldots, l_M)\}$ *is a GCL of* U, *and* $(i_1, \ldots, i_N)$ *is an* N*-tuple of elements of* $\{1, \ldots, M\}\}$.

COROLLARY 6. $S_R \neq \varnothing$ *iff there exists a GCL of* U.

COROLLARY 7. S_R *is symmetric*; *in other words, if* $j_1, \ldots, j_N$ *is any permutation of* $i_1, \ldots, i_N$, *and* $(u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N})$ *is* S_R, *then so is* $(u_{j_1}, l_{j_1}, \ldots, u_{j_N}, l_{j_N})$.

Corollary 6 implies that the problem of reducing R to S_R is itself NP-complete. Indeed, to determine if a graph G is k-colorable, we could create the associated relation R and then reduce R to S_R, where S_R is nonempty iff G is k-colorable; thus the reduction of R to S_R must be an NP-complete problem.

However, simple algorithms do exist for eliminating some parts of R that are not in S_R, as we shall next see.

Figure 6 shows the S_R that corresponds to the R of Fig. 1. The (trivial) search tree for the GCL constrained to contain the interpretation $1-a$ is shown in Fig. 7.

The following corollary to Proposition 5 provides a criterion for identifying $2N$-tuples in R that do not belong to S_R:

COROLLARY 8. *For all* $(u'_1, l'_1, \ldots, u'_N, l'_N) \in S_R$, *and all* $u'_{N+1}, \ldots, u'_p \in U$, *there exist labels* $l'_{N+1}, \ldots, l'_p$ *such that* $(l'_1, \ldots, l'_p)$ *is a consistent labeling of* $(u'_1, \ldots, u'_p)$.

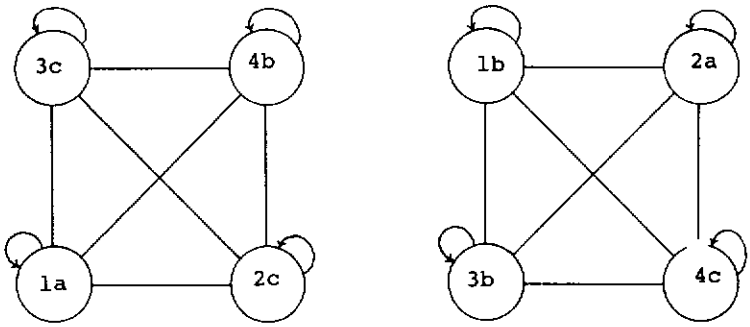


Fig. 6. The reduced relation of Fig. 2.

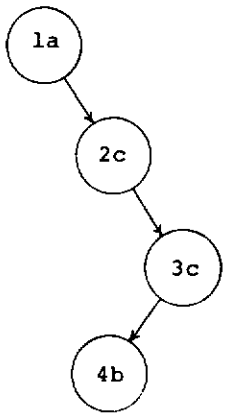


Fig. 7. The search required to find all globally consistent labelings of Fig. 6 with unit 1 constrained to be label a.

This corollary suggests one way of defining a process for "reducing" R—namely, we can keep only those $2N$-tuples that are "extendible" to consistent labelings of P-tuples for some $P > N$. Specifically, for any $P > N$, let us define

$$\Phi_P(R) = \{(u'_1, l'_1, \ldots, u'_N, l'_N) \in R \mid \text{for all } u'_{N+1}, \ldots, u'_P \in U, \text{ there exist } l'_{N+1}, \ldots, l'_P \in L \text{ such that } (l'_1, \ldots, l'_P) \text{ is a consistent labeling of } (u'_1, \ldots, u'_P)\}.$$

PROPOSITION 9. *$R' \subseteq R$ implies $\Phi_P(R') \subseteq \Phi_P(R)$.*

PROPOSITION 10. *$\Phi_P(S_R) = S_R$.*

Proof. This is clear, since the $2N$-tuples in S_R are all defined by GCL's of U. ■

By the same argument we have

PROPOSITION 11. *$S_R \subseteq \Phi_P(R) \subseteq R$.*

Induction on Proposition 11 gives us

COROLLARY 12. *$S_R \subseteq \Phi_P^K(R)$ for all K.*

Note that since R is finite, the sequence $R \supseteq \Phi_P(R) \supseteq \Phi_P^2(R) \supseteq \cdots$ must terminate, i.e., for some K we must have $\Phi_P^K(R) = \Phi_P^{K+1}(R) = \cdots$. Thus repeated application of Φ_P eventually gives rise to a reduced relation $\Phi_P^K(R)$ that is stable under Φ_P and that contains S_R.

If $P \gg N$, it is difficult to determine which $2N$-tuples are in $\Phi_P(R)$, since many possible $(P-N)$-tuples $u'_{N+1}, \ldots, u'_P$ must be considered; hence the Φ_P process for reducing R is easiest to apply for P close to N. Of course, the process is also less stringent for small P, since evidently $P \leqslant Q$ implies $\Phi_P(R) \supseteq \Phi_Q(R)$.

Define πR, the *projection of* R, as the set of $(u, l) \in U \times L$ that are pairs of terms in $2N$-tuples belonging to R.

PROPOSITION 13. *Let $R = \Phi_P(R) \neq \varnothing$, and let $(u'_1, l'_1) \in \pi R$. Then for all $u'_2, \ldots, u'_N$ in U, there exist $l'_2, \ldots, l'_N$ in L such that $(u'_1, l'_1, \ldots, u'_N, l'_N) \in R$.*

Proof. Since $(u'_1, l'_1) \in \pi R$, there exists a $2N$-tuple $(u''_1, l''_1, \ldots, u''_N, l''_N) \in R$ such that, for some n, we have $(u'_1, l'_1) = (u''_n, l''_n)$. Without loss of generality, we may assume that $(u'_1, l'_1) = (u''_1, l''_1)$. Thus $(u'_1, l'_1, u''_2, l''_2, \ldots, u''_N, l''_N) \in R = \Phi_P(R)$. By definition of $\Phi_P(R)$, there exists an l'_2 such that $(l'_1, l'_2, l''_2, \ldots, l''_N)$ is a consistent labeling of $(u'_1, u'_2, u''_2, \ldots, u''_N)$; hence $(u'_1, l'_1, u'_2, l'_2, u''_2, l''_2, \ldots, u''_N, l''_N) \in R$. Repeating the same argument, we can show successively that there exist $l'_3, \ldots, l'_N$ such that $(u'_1, l'_1, \ldots, u'_N, l'_N) \in R$. ■

Another approach to "reducing" R is to identify (unit, label) pairs that are extendable to consistent labelings, and eliminate all $2N$-tuples that do not consist of such pairs. Specifically, for any $T \subseteq U \times L$, and any $P > N$, let us define

$$\Psi_P(T) = \left\{(u,l) \in T \,\middle|\, \begin{array}{l}\text{for all } u_1,\ldots,u_{P-1} \in U, \text{ there exist} \\ l_1,\ldots,l_{P-1} \in L \text{ such that } (u_n,l_n) \in T,\ 1 \leqslant n \leqslant P-1, \text{ and} \\ (l_1,\ldots,l_{P-1},l) \text{ is a consistent} \\ \text{labeling of } (u_1,\ldots,u_{P-1},u)\end{array}\right\}.$$

PROPOSITION 14. *$T' \subseteq T$ implies $\Psi_P(T') \subseteq \Psi_P(T)$.*

PROPOSITION 15. *$\Psi_P(\pi S_R) = \pi S_R$.*

PROPOSITION 16. *For all $T \subseteq U \times L$ we have $\Psi_P(T) \subseteq \pi R$.*

Proof. Any $(u,l) \in \Psi_P(T)$ is part of a consistent labeling; any N-tuple of pairs in a consistent labeling is in R; hence each of the pairs is in πR. ■

COROLLARY 17. *For all T satisfying $\pi S_R \subseteq T \subseteq U \times L$ we have $\pi S_R \subseteq \Psi_P(T) \subseteq \pi R$.*

Proof. Propositions 14, 15, and 16. ■

COROLLARY 18. *For all such T and all k we have $\pi S_R \subseteq \Psi_P^k(T) \subseteq \pi R$.*

Note that since T is finite, the sequence $T \supseteq \Psi_P(T) \supseteq \Psi_P^2(T) \supseteq \cdots$ must terminate, i.e., for some k we must have $\Psi_P^k(T) = \Psi_P^{k+1}(T) = \cdots$. Thus repeated application of Ψ_P to an $T \supseteq \pi S_R$ eventually yields a set of pairs $\Psi_P^k(T)$ that is stable under Ψ_P and that still contains πS_R.

By Propositions 14 and 16 we also have

COROLLARY 19. *$\Psi_P^k(T) \subseteq \Psi_P^{k-1}(\pi R)$ for all k.*

COROLLARY 20. *If $\Psi_P(T) = T$, then $T \subseteq \Psi_P^k(R)$ for all k.*

The operations Ψ_P and Φ_P are related by

PROPOSITION 21. *If $R = \Phi_P(R) \neq \varnothing$, then $\Psi_P(\pi R) = \pi R \neq \varnothing$.*

Proof. Let $(u,l) \in \pi R$. By Proposition *13*, (u,l) can be extended to a $2N$-tuple in $R = \Phi_P(R)$. By definition of Φ_P, this $2N$-tuple can be extended to a consistent labeling of some P-tuple, whose pairs are thus all in πR; hence $(u,l) \in \Psi_P(\pi R)$. ■

Figure 8 helps to illustrate the comparative power of Ψ_P and Φ_P in reducing R towards the minimal constraint. Consider the application of Ψ_3 and Φ_3 to

(u_2, u_3)	(l_2, l_3)
(2,3)	(a,a)
(3,4)	(a,a),(a,b)
(2,4)	(a,a)

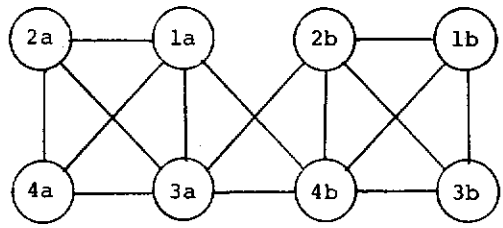


Fig. 8.

the network of binary constraints shown in Fig. 8. We will first focus attention on the node-label pair $(1,a)$ to determine if $(1,a) \in \Psi_3(\pi R)$. From the definition of Ψ_3, for every pair of units (u_2,u_3), we must find a pair of labels (l_2,l_3) such that $(1,a),(u_2,l_2),(u_3,l_3)$ form a triangle in Figure 8. The table in Fig. 8 lists the (l_2,l_3) for each pair of units (u_2,u_3) (ignoring the trivial pairs that include $u_2=1$). Since every pair (u_2,u_3) has a label pair (l_2,l_3) that, along with $(1,a)$, forms a triangle in Fig. 8, we have $(1,a) \in \Psi_3(\pi R)$.

Now consider the application of Φ_3 to R. Let us specifically ask if $((1,a),(4,b)) \in \Phi_3 R$. From the definition of Φ_3, for every unit u_3 we must find an l_3 such that $(1,a),(4,b),(u_3,l_3)$ form a triangle in R. Notice that for $u_3=2$, we can find no such l_3. Therefore $((1,a),(4,b)) \notin \Phi_3 R$. Similarly, we can see that $((2,b),(3,a)) \notin \Phi_3 R$, since there is no suitable l_3 for $u_3=1$.

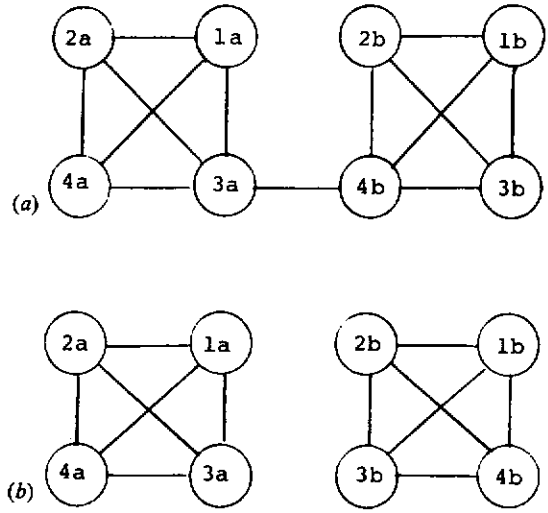


Fig. 9.

Thus, although πR is a fixed point of Ψ_3 [since every (u,l) is a part of a GCL], it is not a fixed point of Φ_3 (since $R \neq S_R$ and this can be detected by Φ_3). Figure 9(a) shows $\Phi_3(R)$, and Fig. 9(b) shows $\Phi_3^2(R)$.

We can relate the notions of path consistency and arc consistency to fixed points of Φ_3 and Ψ_2, respectively.

PROPOSITION 22. *If $R \subseteq (U \times L)^2$ and $\Phi_3(R) = R$, then R is path-consistent.*

Proof. Suppose $\Phi_3(R) = R$. Let $r = (u_i, l_i, u_j, l_j) \in R$. Then for all u_k, there is an l_k such that (l_i, l_j, l_k) is a consistent labeling of (u_i, u_j, u_k). But then (u_i, l_i, u_k, l_k) and (u_k, l_k, u_j, l_j) are in R. ■

Note that path consistency is not enough to ensure that $R = \Phi_3(R)$, since (u_i, l_i, u_k, l_k) and (u_k, l_k, u_j, l_j) in R do not imply that (u_i, l_i, u_i, l_i) or (u_k, l_k, u_i, l_i) is in R. The first condition can be guaranteed by introducing the notion of node consistency (after Mackworth [1]), and the second by demanding that R be symmetric.

DEFINITION. $R \subseteq (U \times L)^2$ is node-consistent iff $(u,l) \in \pi R$ implies $(u,l,u,l) \in R$.

We can now state

PROPOSITION 23. *If $R \subseteq (U \times L)^2$ is symmetric and node-consistent, then R is path consistent iff $R = \Phi_3(R)$.*

Symmetry is not necessary to relate arc consistency to Ψ_2:

PROPOSITION 24. *Let $R \subseteq (U \times L)^2$ be node-consistent. Then R is arc-consistent iff $\pi R = \Psi_2(\pi R)$.*

Proof. Suppose R is arc-consistent. Let $(u,l) \in \pi R$. Then for every u' there is an l' such that $(u,l,u',l') \in R$. Thus l, l' is a consistent labeling of (u,u') and $(u',l') \in \pi R$. By definition of Ψ_2 we have $(u,l) \in \Psi_2(\pi R)$. Therefore $\pi R \subseteq \Psi_2(\pi R)$, and since $\Psi_2(\pi R) \subseteq \pi R$, this proves that $\pi R = \Psi_2(\pi R)$.

Conversely, suppose $\pi R = \Psi_2(\pi R)$. Then $(u,l) \in \pi R$ implies that for every u' there is an l' such that $(u',l') \in \pi R$ and $(u,l,u',l') \in R$. But since R is node-consistent, we have $(u,l,u,l) \in R$. Hence R is arc-consistent. ■

In the next section we will see how Φ and Ψ can be incorporated into search procedures that compute the GCLs of the relation R.

4. THE USE OF REDUCTION OPERATORS IN SEARCH

Both Ψ and Φ can be directly incorporated into a backtracking procedure which searches the GCLs of a relation R. Each step in the backtracking process places a node in a search tree corresponding to the instantiation of a

particular set of units as a specific set of labels. These instantiations must be checked for consistency with the other instantiations on the path from the root leading to that new node. If the instantiations are consistent with the previous assignments, then either Ψ or Φ can be applied at that node in the search tree. This should reduce the number of labels that an uninstantiated variable may assume, contingent on the variable assignments made along the path.

Suppose that a son, N_k, is added to node N_i in the search tree by instantiating the variables $u_i,\ldots,u_k$ to the labels $l_i,\ldots,l_k$ respectively. Traditional backtracking algorithms would check for the consistency of this instantiation with the other instantiations along the path from the root to N_i. If units $u_1,u_2,\ldots,u_n$ were instantiated to labels $l_1,\ldots,l_n$ on the path from the root to N_k, then the backtracking procedure would check that $(u_{i_1},l_{i_1},\ldots,u_{i_N},l_{i_N})\in R$ for all $u_{i_1},\ldots,u_{i_N}$ in $\{u_1,\ldots,u_n\}$. However, the backtracking does not allow the instantiations of these units to constrain the future instantiation of other units further down in the search tree. Incorporating Φ or Ψ into the search procedure provides such a facility. We will first describe how Ψ can be included in the backtracking procedure and then consider Φ. The description willl include an example of the search process.

Every node N_i in the search tree can be represented as an ordered pair (I_i,E_i), where I_i is the set of instantiations made along the path to N_i and E_i is the set of possible extensions to I_i. At the root, e.g., $I_i=\varnothing$ and $E_i=\pi R$. Adding a son N_j to N_i involves choosing a $(u,l)\in E_i$, and setting a temporary $I_j'\equiv I_i\cup\{(u,l)\}$ and $E_j'\equiv E_i-\{(u,l')|l'\in L\}$. So far, this is equivalent to the standard backtracking algorithm. However, we will apply Ψ to $I_j'\cup E_j'$, and finally set $I_j=I_j'\cap\Psi^k(I_j'\cup E_j')$ and $E_j\cap\Psi^k(I_j'\cup E_j')$, where k is large enough so that Ψ reaches its fixed point. Then, if $I_j=\varnothing$, the search can be discontinued below N_j.

As an example of how the process works consider Fig. 10 and the relation R displayed in Fig. 2. Figure 10 shows the partial development of the search tree. At N_1 unit 1 has been assigned label a, which yields $I_1'=\{(1,a)\}$ and $E_1'=\pi R-\{(1,a),(1,b),(1,c)\}$. Applying Ψ_2 to $I_1'\cup E_1'$ greatly reduces the size of E_1 relative to E_1'. This means that, potentially, the depth of the search below N_1 will be much less than it would have been if Ψ_2 had not been applied. Extending N_1 to N_2 by instantiating unit 2 to label a, and then applying Ψ_2, causes $I_2=E_2=\varnothing$, so that search may be discontinued below N_2 (compare Figure 4). Notice that there is an increase in storage complexity associated with including Ψ into the backtracking—namely, the addition of E_1 at each node in the search tree.

To incorporate Φ into the search, we associate a subrelation R_i of R at each node. R_i is the set of constraints (i.e., $2N$-tuples) from R that are compatible with the instantiation of units along the path to N_i. Nodes are now ordered pairs (I_i,R_i), with E_i implicitly defined by I_i and R_i—i.e., $E_i=\pi R_i-\{(u,l)|u$ is instantiated in $I_i\}$.

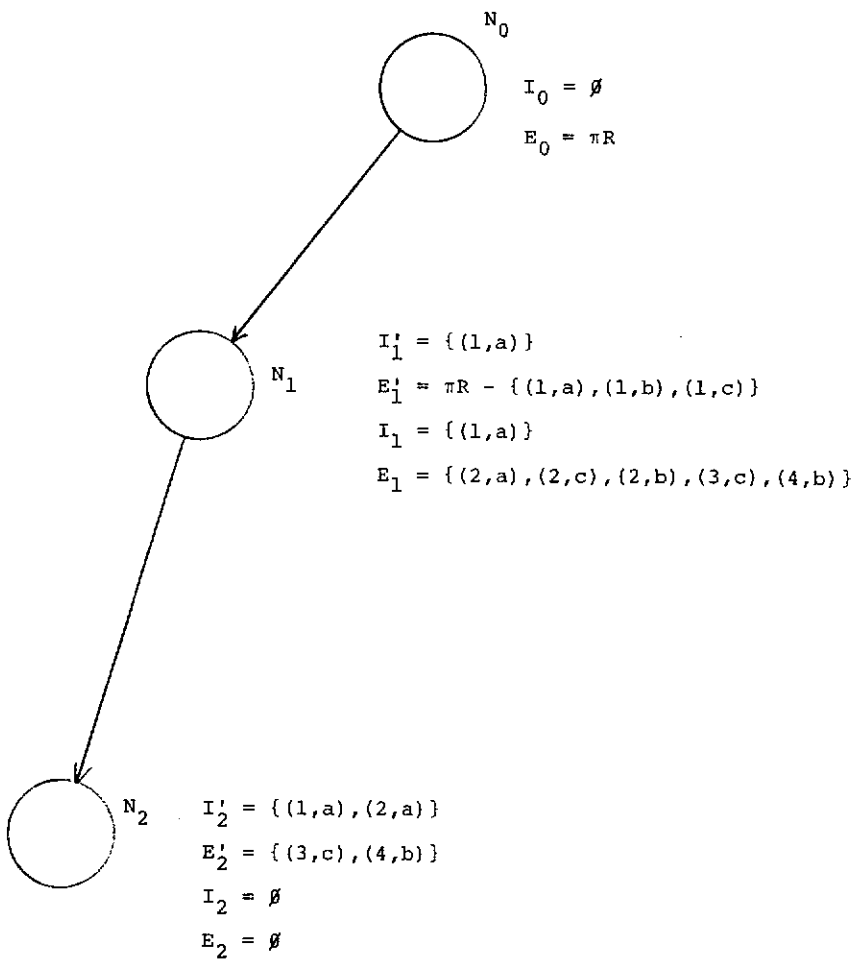


Fig. 10.

The search proceeds as follows: a unit-label pair (u,l) is chosen from E_i, and $I_j' = I_i \cup \{(u,l)\}$. Next, $R_j' = R_i - \{(u_1,l_1,\ldots,u_N,l_N) | u_i = u$ and $l_i \neq l$ for some $i\}$—i.e., all $2N$-tuples in R_i that contain unit u but do not assign label l to it are removed from R_i to form R_j'. Then $R_j = \Phi^k(R_j')$ is chosen, where k is large enough so that the fixed point of Φ is attained. Again the search can be discontinued below N_j if $I_j = \varnothing$.

Figure 11 shows the result of including Φ_3 in the search for the GCLs of Fig. 2. N_1 is obtained by assigning label a to unit 1, and this results in the R_1' shown in Fig. 11 [erase nodes $(1,b)$ and $(1,c)$ from Fig. 2 and delete all arcs

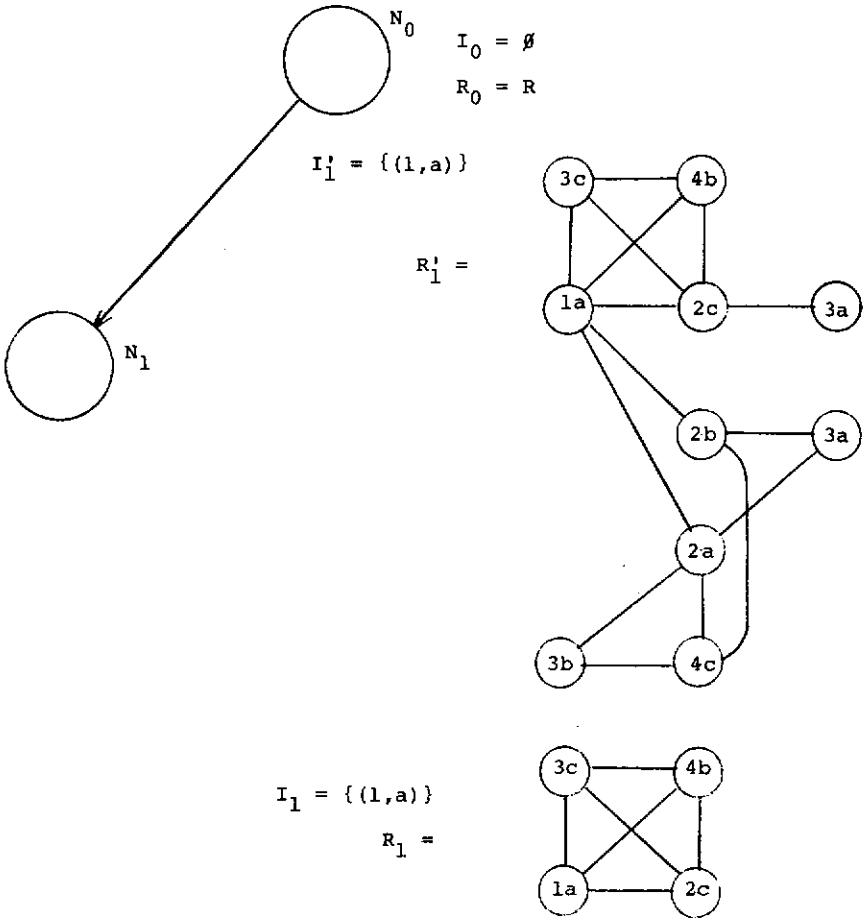

N_0
$I_0 = \emptyset$
$R_0 = R$
$I_1' = \{(1,a)\}$
$R_1' =$
N_1
3c
4b
1a
2c
3a
2b
3a
2a
3b
4c
$I_1 = \{(1,a)\}$
$R_1 =$
3c
4b
1a
2c

Fig. 11.

that impinged on them]. Applying Φ_3 (one iteration) to R_1' yields R_1. Note that we would have achieved much the same results by applying Ψ_3 to $\pi R_1'$, but that in general this will not be the case (see again Fig. 8). Incorporating Φ into the search requires that we store R_i at each node N_i in the search tree; this is, of course, a greater storage requirement than that demanded by Ψ.

5. COMPUTATIONAL COSTS AND STORAGE REQUIREMENTS

The choice of whether to use Φ or Ψ in the search for GCLs will be dictated by both their storage and computational requirements. We will examine the computational requirements by analyzing algorithms that compute Ψ and Φ. It should be pointed out in advance that since we cannot establish the optimality of these algorithms, any conclusions drawn on the basis of their analysis would, of course, be invalidated by the exhibition of more efficient algorithms.

We first consider the following algorithm for computing $\Phi_P(R)$:

ALGORITHM CR

For each $(u_1, l_1, \ldots, l_N, l_N) \in R$
For each $u_{N+1}, \ldots, u_P \in U$

> *If there exists a consistent labeling* $(l_1', \ldots, l_P')$ *of* $(u_1, \ldots, u_P)$ *with* $l_i' = l'$, $i = 1, \ldots, N$, *then mark* $(u_1, l_1, \ldots, u_N, l_N)$ *as an element of* $\Phi_P R$.

If $\# R = r$, then the cost of Algorithm CR is $rM^{RN}\, C$, where C is the cost of an optimal procedure for deciding if there exists a consistent labeling $(l_1', \ldots, l_P')$ of $(u_1, \ldots, u_P)$ with $l_i' = l_i$, $l = 1, \ldots, N$.

When iterating Algorithm CR, further efficiency can be achieved by noticing that to compute $\Phi_P^k(R)$ it is only necessary to search for consistent labelings of those P-tuples $(u_1, \ldots, u_P)$ such that some relation involving one of the u_i in $(u_1, \ldots, u_P)$ was deleted by the application of Φ_P to $\Phi_P^{k-1} R$. This is captured by the following proposition (and is the generalization of the queuing of nodes described by Mackworth [1]):

PROPOSITION 25. *Let* $(u_1, l_1, \ldots, u_N, l_N) \in \Phi_P^{k-1}(R)$. *Let* $u_{N+1}, \ldots, u_P$ *be such that for all* l, $(u_i, l) \in \pi\big(\Phi_P^{k-2}(R) - \Phi_P^{k-1}(R)\big)$, $i = N+1, \ldots, P$. *Then there is a consistent labeling* $(l_1', \ldots, l_P')$ *of* $(u_1, \ldots, u_P)$ *with* $l_i' = l_i$, $i = 1, \ldots, N$.

Proof. Since $(u_1, l_1, \ldots, u_N, l_N) \in \Phi_P^{k-1}(R)$, for the $u_{N+1}, \ldots, u_P$ of the hypothesis, there exist $l_{N+1}, \ldots, l_P$ such that $(l_1, \ldots, l_P)$ is a consistent labeling of $(u_1, \ldots, u_P)$ in $\Phi_P^{k-2}(R)$. This means that $(u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N}) \in \Phi_P^{k-2}(R)$, $i_j \in \{1, \ldots, P\}$. But since $(u_1, l_1, \ldots, u_N, l_N) \in \Phi_P^{k-2}(R)$, and every $2N$-tuple in $\Phi_P^{k-2}(R)$ containing a component (u_i, l), $i = N+1, \ldots, P$, is in $\Phi_P^{k-1}(R)$, it must be that $(u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N}) \in \Phi_P^{k-1}(R)$—i.e., $(l_1, \ldots, l_P)$ is a consistent labeling of $(u_1, \ldots, u_P)$ in $\Phi_P^{k-1}(R)$. ■

Next, consider the following algorithm to compute $\Psi_P(\pi R)$:

ALGORITHM NR

For each $(u_1, l_1) \in \pi R$ DO
  OK = 1
  For each $(u_2, \ldots, u_N)$ DO
  Let $\mathfrak{L} = \{(l_2, \ldots, l_N) \mid (u_{i_1}, l_{i_1}, \ldots, u_{i_N}, l_{i_N}) \in R,$
    $(i_1, \ldots, i_N)$ *a permutation of* $(1, \ldots, N)\}$
  If $\mathfrak{L} = \emptyset$, *then* OK = 0 *else*
    If for all $u_{N+1}, \ldots, u_P$ *there exist* $l_{N+1}, \ldots, l_P$ *such that* $(l_1, l_2, \ldots, l_N, l_{N+1}, \ldots, l_P)$ *with* $(l_2, \ldots, l_N) \in \mathfrak{L}$ *is a consistent labeling of* $(u_1, \ldots, u_P)$
      Then OK = 1 FI
    FI
    OK = OK * 1
  If (OK .EQ. 0) EXIT FOR OD
  $(u_1, l_1) \in \Psi_P(\pi R)$ *if and only if* OK = 1.

If s is the average size of an $\mathfrak{L}$-set, then the cost of Algorithm NR (ignoring the cost of computing $\mathfrak{L}$) is $(\#\pi R)(M^{N-1})(s)(M^{P-N})C$. Now, it is reasonable to assume that $s \cdot M^{N-1} \approx r/\#\pi R$, since R is symmetric and the average number of elements of R for fixed $(u_1, l_1, u_2, \ldots, u_N)$ was taken to be s. Thus $r \approx (\#\pi R)(M^{N-1})(s)$, and with this assumption the computational cost of Algorithm NR $\approx rM^{P-N}C$, which is the same as the cost of Algorithm CR. Furthermore, we did not take into account the cost of computing the set $\mathfrak{L}$ in Algorithm NR; this would entail some examination of R. If R were stored in a multidimensional binary forest induced by the lexicographic ordering of U and L (Bentley [11]), then we could assume that the cost of computing $\mathfrak{L}$ is proportional to $N \log r$. Thus on the basis of computational cost, there seems to be no reason for choosing operation Ψ_P over operator Φ_P.

The operator Ψ_P requires the storage of E_i at each node N_i of the search tree. Each such E_i can be represented by dM bits of storage, one for each unit-label pair.

Operator Φ_P requires that R_i be maintained at each node N_i. Since $R_i = \Phi_P^k R \subseteq (\pi \Phi_P^k R_i)^N \cap R$, one might think it would be effective to store only $(\pi \Phi_P R)^N$ at each node in the search and restore an "expanded" version of $\Phi_P^k R$ when search returns to a node by computing the previous intersection. Note that Φ_P may be more powerful than Ψ_P, since we know that for k large enough, $\pi \Phi_P^k(R) \subseteq \Psi_P^k(R)$.

It is important to note, however, that the cost of computing R_i' does not balance the cost of computing the $\mathfrak{L}$-sets required by Algorithm CR. R_i' can be

computed by a procedure that requires $(\#\pi R_i)$ $(N \log r)$ operations; for each element (u,l) of πR_i we must thread through the multidimensional binary tree of elements of R whose first unit-label pair is (u,l) and find all those constraints composed only of elements from πR_i. This cost adds to the cost of applying Algorithm CR to R_i'. However, the cost of computing an $\mathfrak{L}$-set was shown in Sec. 4 to multiply the cost of Algorithm NR.

6. CONCLUDING REMARKS

Some general methods of reducing the size of an order-N compatibility relation have been defined, and properties of these procedures have been derived. These polynomial-time methods can be incorporated into a backtracking procedure in order to reduce the expected (although not necessarily the worst-case) computational cost of searching for compatible labelings. Thus these methods should be useful in helping to solve the constrained labeling problem in many practical situations.

REFERENCES

1. A. Mackworth, Consistency in networks of relations, *Artificial Intelligence* **8**, 99–118 (1977).
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
3. U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Information Sci.* **7**, 95–132 (1974).
4. W. L. Waltz, Understanding line drawings of scenes with shadows, in *The Psychology of Computer Vision* (P. H. Winston, Ed.), McGraw-Hill, New York, 1975.
5. A. Rosenfeld, Networks of automata: some applications, *IEEE Trans. on Systems, Man and Cybernetics*, *5*, 380–383, 1975.
6. A. Rosenfeld, R. Hummel and S. Zucker, Scene labeling by relaxation operations, *IEEE Trans. Systems, Man and Cybernet.* **6**, 420–433 (1976).
7. J. Gaschnig, A constant satisfaction method for inference making, 12th Annual Allerton Conf. on Circuit and System Theory, Univ. of Illinois, 1974.
8. R. M. Haralick and J. Kartus, Theory of arrangements, Univ. of Kansas Center for Research, Lawrence, KS, 1976.
9. H. Barrow and J. M. Tenenbaum, MSYS: A system for reasoning about scenes, Stanford Research Institute AI Center Tech. Note 121, 1976.
10. L. Davis, Shape matching using relaxation techniques, Univ. of Maryland Computer Science Center Tech. Report 480, College Park, MD, 1976.
11. J. L. Bentley, Multidimensional binary search trees used for associative searching, *Comm. ACM* **18**, 509–516 (1975).

Received December 1977