ISE-TR-86-55

CONSISTENT LABELING ALGORITHM USING THE DYNAMIC PROGRAMMING CONCEPT

by

Seiichi Nishihara

Tsunemichi Shiozawa

Katsuo Ikeda

February 18, 1986

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

Consisten Labeling Algorithm Using the Dynamic Programming Concept

Seiichi NISHIHARA

Tsunemichi SHIOZAWA

Katsuo IKEDA

Institute of Information Sciences and Electronics
University of Tsukuba
Sakura-mura, Niihari-gun
Ibaraki 305, Japan

Abstract

Being given an object consisting of many subparts and their locally legal interpretations, problems of finding totally consistent interpretations are found in many areas, examples of which are image analysis and artificial intelligence. Search problems of this kind are called consistent labeling problems (CLPs). There are two well known principal strategies for the CLP: the depth-first approach typified by backtracking and the breadth-first approach typified by constraint propagation. This paper proposes another approach that makes use of the dynamic programming concept, whose basic idea is reducing the given problem into several smaller subproblems and eliminating variables one by one.

1. Introduction

1.1 Consistent Labeling Problem (CLP)

A CLP is represented by a quadruple (U,L,T,R), where U is a set of units {1,...,M}, and L is a set of labels. Labels are usually possible interpretations, meanings or values being assigned to the units; T is a set of tuples of units, i.e., $T \subseteq \bigcup_{1 \le i \le M} U^i$. Each tuple t in T directs the units composing t to mutually constrain one another. And all of the permitted or legal labelings for t are given explicitly by a #t-ary relation of labels R_t ($\subseteq L^{\#t}$), which is called a label constraint relation; #t is the dimension of tuple t. And, $R=\{R_t \subseteq L^{\#t} \mid t \in T\}$.

Solving a CLP is to find all consistent labelings $\lambda=(l_1,\ldots l_M)$ of units (1,...,M) satisfying $\forall t\ (\in T)\ (\lambda(t) \in R_t)$, where $\lambda(t)$ is the projection $(l_{u1},\ldots,l_{u\#t})$ of λ on $t=(u_1,\ldots,u_{\#t})$. Below, we give an example of a CLP.

Example 1

U={1,...,9}, L={a,...,j},

T={t_1,...,t_7},

t_1=(1,2,4), t_2=(1,2,3), t_3=(2,3,4,5),

t_4=(5,8), t_5=(4,6,7), t_6=(6,7,8), t_7=(7,8,9),

R={R_1,...,R_7},

R_1={(a,b,c),(b,a,c),(b,a,e)},

R_2={(b,a,b),(a,b,c),(a,b,a)},

R_3={(a,b,c,d),(a,b,e,d),(b,c,c,d)},

$R_4=\{(d,h),(e,i)\}$,

$R_5=\{(c,d,f),(d,c,e),(c,f,g)\}$,

$R_6=\{(f,g,h),(d,f,h),(g,h,i)\}$,

$R_7=\{(f,h,j),(d,f,j),(g,h,h)\}$.

The above definition of CLP is a generalized version of the one given by Haralick and Shapiro[1] in the sense that it admits many multiple constraint relations whose dimensions are not necessarily the same each other.

1.2 Constraint Network

The constraint network providing an equivalent expression of a CLP is defined by an undirected graph G=(V,E):

$$V=\{(t_i,R_i)\mid t_i\in T,\ R_i\in R\},$$

$$E=\{(v_i,v_j)\mid s(t_i)\cap s(t_j)\neq\phi,$$

$$v_i=(t_i,R_i),v_j=(t_j,R_j),\ v_i,v_j\in V\}$$

where $s(t_i)=\{u_1,\ldots,u_k\}$, letting $t_i=(u_1,\ldots,u_k)$.

As is shown above, a pair (t_i,R_i) called 'constraint-pair', is assigned to each vertex v_i. Thus the term constraint-pair may often be used to indicate the corresponding vertex. Our definition of constraint networks differs from the conventional ones[2,3] in which each vertex corresponds to a single unit and only binary relations are permitted. In the constraint networks proposed here, each tuple in T corresponds to a vertex; thus,

relations with arbitrary dimensions can be expressed in a straightforward manner. Each edge represents a local constraint in that the labels associated with the common units appearing in two vertices (i.e., tuples) connected to each other by an edge, are to be equivalent.

Fig. 1 shows the constraint network equivalent to the CLP of Example 1.

****
* Fig. 1 *
****

1.3 Redundant Units

As is seen from Fig. 1, unit 9 appears only in v_7 because t_7 is (7,8,9), meaning only unit 7 and unit 8 share constraints with unit 9. Thus, by saving relation R_7, unit 9 can be eliminated from the network to reduce the problem to a slightly smaller one. Units such as 9 are called 'redundant'. Next, assume two vertices v_1 and v_2 are joined into a single one, v_{12}; then, unit 1 newly becomes redundant (see Fig. 2) and can be eliminated this time. The idea that redundant units can be eliminated gradually to reduce the given problem into smaller ones is similar to the basic strategy of dynamic programming.

****
* Fig. 2 *
****

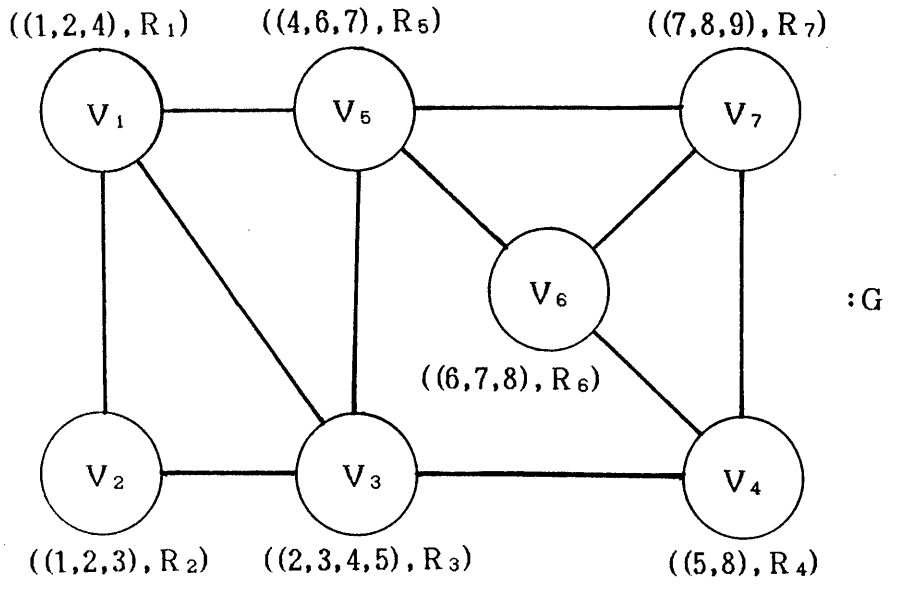

Fig. 1 The constraint network equivalent to Example 1.

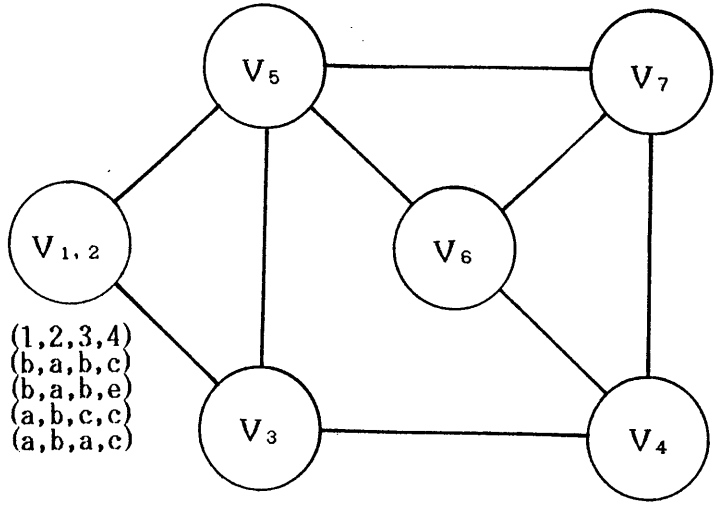

Fig. 2 The constraint network derived after joining v_1 and v_2 of Fig. 1.

2. A Method Using Invasion

2.1 Invasion

Consider the following two cases of joining sequences applied to Fig. 2: One is to join v_{12} and v_5, and the other is to join v_{12} and v_3. The constraint network resulting from each of these two join operations is shown in Fig. 3(a) and (b). In Fig. 3(b), units 2 and 3 become redundant because those units do not appear in any other vertices except v_{123}, while no redundant unit is produced in Fig. 3(a).

****
* Fig.3(a) *
****

****
* Fig.3(b) *
****

Thus our example shows that a sequence of join operations fully determines when each unit becomes redundant. This sequence is uniquely expressed by a series of subgraphs of the given constraint network, called an invasion after [2], which is defined as follows. An 'invasion' of a given network G is a sequence of induced subgraphs[4] $G_1, G_2, \ldots, G_{|T|}$, where G_i is composed of i vertices and $V(G_i) \subsetneq V(G_{i+1})$ for $i=1,\ldots,|T|-1$. $|T|$ represents the size of set T, i.e. the number of vertices, since to each vertex v_i of G a constraint-pair $(t_i(\in T), R_i(\in R))$ is

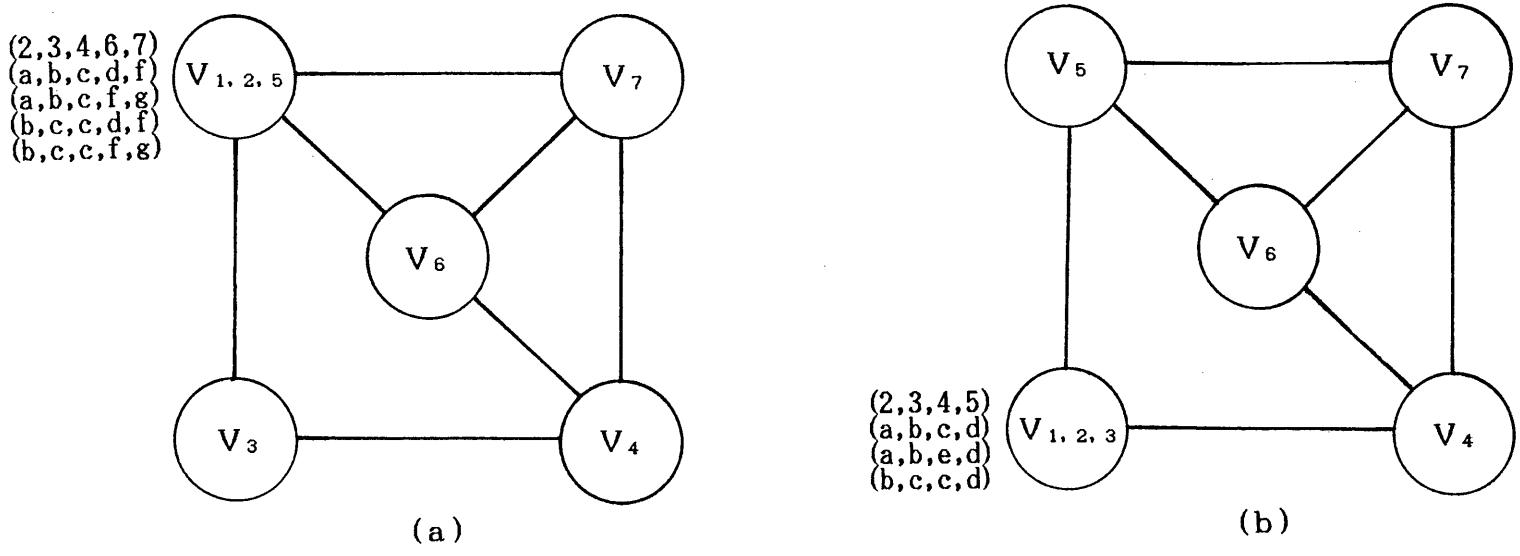

Fig. 3 The network after joining (a) v_{12} and v_5, and (b) v_{12} and v_3.

assigned. In particular, $G_{|T|}$=G. Fig. 4 shows an example of an invasion of the network given in Fig. 1.

****
* Fig. 4 *
****

In Fig. 3(b), units 4 and 5 appear in some vertices other than v_{123}, since they are non-redundant components of the tuple assigned to vertex v_{123}. Unit 4, for example, appears as a component of tuple t_5(=4,6,7)) of vertex v_5. The set of such non-redundant units is called the 'front', whose concrete definition is as follows:

Let us denote the front of G_i, which is a subgraph of G, as $f(G_i)$. Then,

$$f(G_i) \stackrel{def}{=} \Big(\bigcup_{v(=(t,R))\in V(G_i)} s(t) \Big) \cap \Big(\bigcup_{v(=(t,R))\in V-V(G_i)} s(t) \Big),$$

where $V(G_i)$ and V are the sets of vertices of graph G_i and the original graph G, respectively. As a special case, $f(G_0)$ is empty, since G_0 is naturally assumed to be an empty graph. Notice that the above definition of front can be applied to any subgraph of G.

By using the above definition of fronts, the set of redundant units which occurs after joining the vertex of $V(G_i)$-$V(G_{i-1})$ and its adjacent vertices is expressed as $f(G_{i-1})$-$f(G_i)$. Let us call this the redundant unit set of G_i and denote it as

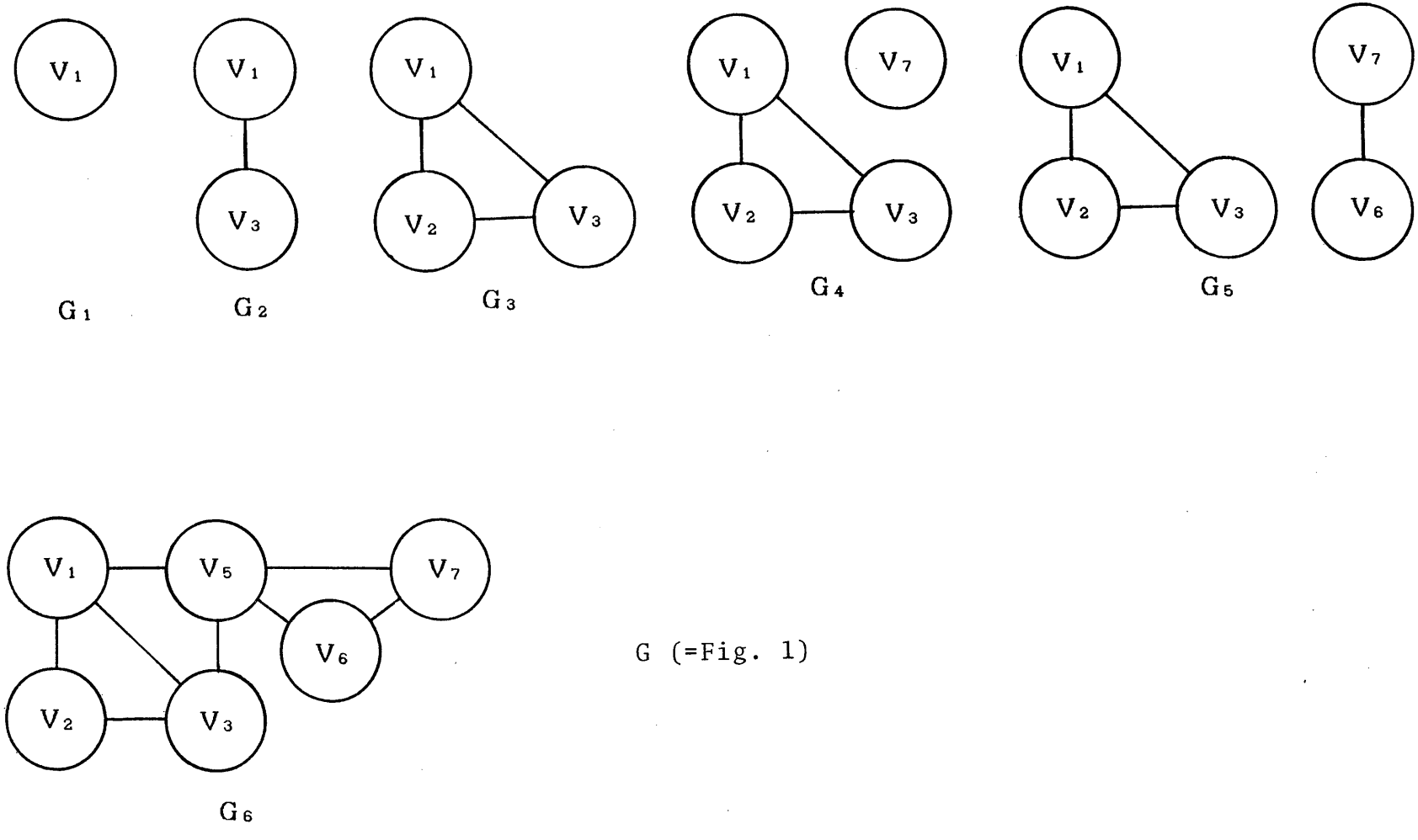

Fig. 4 An example of an invasion of network G shown in Fig. 1.

$r(G_i)$.

Let $G_i^{(1)},\ldots,G_i^{(c)}$ be connected components of the i-th subgraph $G_i(\subseteq G)$ of an invasion. The 'front length'[3] of G_i, denoted as ϕ_i, for $1\leq i\leq n$ is defined as follows:

$$\phi_i = \begin{cases} \max\limits_{1\leq j\leq c} |f(G_i^{(j)})| & \text{, if } r(G_i) \text{ is empty,} \\ 0 & \text{, if } r(G_i) \text{ is non-empty.} \end{cases}$$

This definition effectively gives a meaningful value only in the case where some redundant units are produced. For each i ($1\leq i\leq |T|$), if at least one redundant unit is produced with respect to G_i, i.e. if $r(G_i)$ is non-empty, the vertex in $V(G_i)-V(G_{i-1})$ is called a 'merging-point'.

In the example of Fig. 4, all of the G_i's in which a merging-point appears are listed below with their merging-points, redundant units, fronts and front lengths:

	G_3	G_4	G_6	G_7
merging-point	v_2	v_7	v_5	v_4
redundant units $r(G_i)$	{1,2,3}	{9}	{4,6,7}	{5,8}
front $f(G_i)$	{4,5}	{4,5,7,8}	{5,8}	empty
front length ϕ_i	2	2	2	0

2.2 An Algorithm Using an Invasion

Given an invasion $\{G_i\}$, $i=1,\ldots,|T|$ of a constraint network G, we describe a consistent labeling algorithm which is composed

of two main procedures: the forward and the backward phases.

The forward phase generates a solution graph which keeps the partial solutions derived by each join operation and their interrelationship. A join operation is actually invoked at each merging-point, where at least one unit becomes redundant, and includes finding all label tuples for the front which are consistent with one or more label tuples for the redundant units. Thus, a join operation is applied to two or more vertices. Every pair consisting of 1) a label combination for the redundant units, which is found in the merging-point, and 2) a possible combination of labels for the front units is checked if it satisfies the local constraint existing between the merging-point and its adjacent vertices. The join operation also replaces all vertices selected for the join with a single vertex.

The forward phase is described as follows:

```
procedure Forward;
    initialize solution graph with only two nodes, S and E;
    for every merging-point v(i) (∈V(Gi)) do
        begin
            join:find all locally consistent labelings for
              r(Gi) and f(Gi(p));
            replace Gi(p) with a single vertex eliminating
              r(Gi);
            extend the solution graph by adding arcs
              (associated with labelings for r(Gi)) and nodes
              (associated with labelings for f(Gi(p))
```

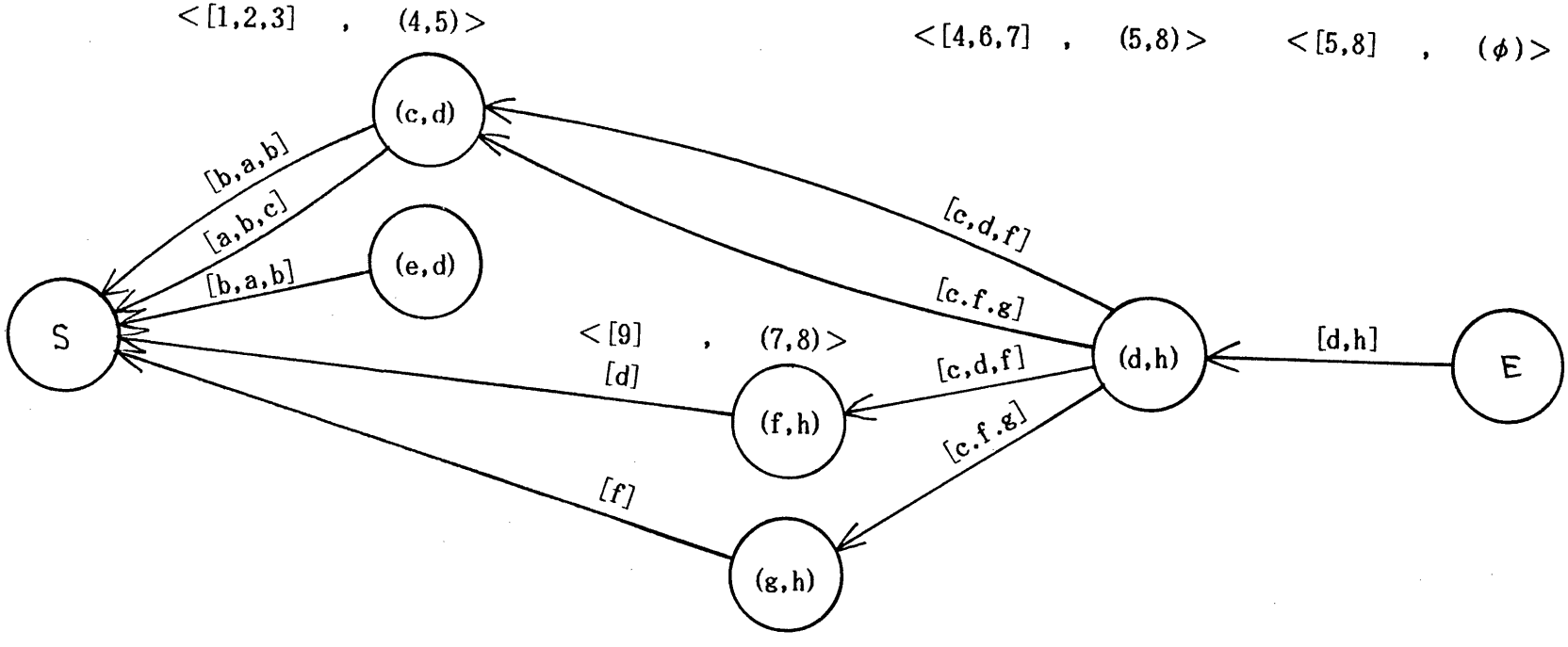

Fig. 5 The solution graph produced by the forward procedure applied to G in Fig. 1 using the invasion in Fig. 4.

{Notice: if $f(G_i^{(p)})=\phi$, add only the arcs
and connect them to node E}

end.

The forward procedure results in a completed solution graph. As an example, Fig.5 is a solution graph derived by applying the above procedure to the constraint network in Fig. 1 using the invasion in Fig. 4.

****
* Fig. 5 *
****

The backward phase constructs all the final solutions for the given constraint network, i.e. the given CLP, by tracing the solution graph backward from node E to node S in the reverse order of the generation sequence of nodes.

Starting from E in Fig. 5, the arc [d,h] and the node (d,h) are first visited. As a general rule for tracing, if there are two or more arcs emanating from the same node and being assigned the same labelings, all of them must be traced together. Thus one of the traces that leads to a final solution is

$$E \rightarrow [d,h] \rightarrow (d,h) \rightarrow [c,d,f] \begin{cases} \nearrow (c,d) \rightarrow [b,a,b] \rightarrow S \\ \searrow (f,h) \rightarrow [d] \rightarrow S \end{cases}$$

where [...] and (...) represent labelings assigned to an arc and a node, respectively. The final solution found by the above tracing is

units (5 8 4 6 7 1 2 3 9)

solution (d h c d f b a b d)

2.3 An Analysis of Computational Complexity

Since tracing a solution graph is straightforward, the backward phase requires time and space proportional to the number of solutions. However, the efficiency of the forward phase is affected by the invasion used. In the join step, a part of the forward procedure, the number of candidate labelings for $r(G_i)$ does not exceed the size, k, of label constraint relation R_i letting $v_i=(t_i,R_i)$. Assume all of the sizes of the label constraint relations in R is equal to k. The maximum number of possible combinations of labels for $f(G_i)$ is $(\min\{k,|L|\})^{|f(G_i)|}$. Therefore, the total number of candidate labelings to be checked by the forward procedure is given as

$$\sum_{i\in I} k\cdot(\min\{k,|L|\})^{\phi_i} \quad ,$$

where I is the set of indices of all merging-points. This shows that the front lengths of the given invasion affect the total computational efficiency in term of both time and space.

3. Optimal Invasion

3.1 Several Characteristics

As a first approximation, let us define the front length, Φ, of an invasion $\{G_i\}$, $i=1,\ldots,|T|$, as $\Phi=\max_i \phi_i$, which gives the order of the computational upper bound of the forward phase. Let τ_u be a subset of constraint-pairs where

$$\tau_u=\{(t_i,R_i)\ (\in V)\ |\ u\in s(t_i)\}$$

for any unit u in U.

Lemma 1. An arbitrary invasion can uniquely be determined by an appropriate sequence of constraint-pairs, and vice versa.

Lemma 2. Let $\rho(\subseteq U)$ be a set of redundant units. Then, there exists a tuple t in T satisfying $\rho\subseteq s(t)$.

Lemma 3. The last subgraph $G_{|T|}$ of an invasion, which is the same as the original constraint network, always contains a merging-point, and further, $f(G_{|T|})=\phi$.

By using Lemma 1, we can denote the front length of the invasion uniquely determined by a given sequence of constraint-pairs $v_1,\ldots,v_{|T|}$, as $\Phi_{<v_1,\ldots,v_{|T|}>}$.

Theorem 4. Assume a sequence of constraint-pairs, $v_1,\ldots,v_{|T|}$, is given. Let v_n be an arbitrary merging-point and $r(G_n)$ be the set

of its redundant units. Let v_m ($1 \leq m \leq n$) be the constraint-pair satisfying that v_{m-1} is the last merging-point which appears in the sequence prior to v_n. If v_n is the first merging-point in the sequence then m=1. Let $w_m,\ldots,w_n$ be a permutation of sequence $v_m,\ldots,v_n$ satisfying $r(G_n) \subseteq s(t_n)$, where t_n is the unit tuple of the constraint-pair w_n. Then, the front length of the invasion determined by the sequence of constraint-pairs whose part $v_m,\ldots,v_n$ is replaced by $w_m,\ldots,w_n$, is equal to the original front length $\Phi_{<v_1,\ldots,v_{|T|}>}$.

Corollary 5 In Theorem 4, any permutation of subsequence $v_m,\ldots,v_{n-1}$ keeps the front length unchanged.

Theorem 6. Let σ be a (non-empty) subset of s(t), where $t(\in T)$ is a tuple in T. Let $\Gamma(\sigma)$ is a set of constraint-pairs as

$$\Gamma(\sigma) = \bigcup_{u \in \sigma} \tau_u .$$

Then,

$$\Phi_{min} \geq \min_{\sigma} |f(<\Gamma(\sigma)>_G)|,$$

where Φ_{min} is the minimum front length of the given constraint network G, and $<\Gamma(\sigma)>_G$ is the subgraph of G induced by $\Gamma(\sigma)$.

3.2 An Algorithm for an Optimal Invasion

An invasion having a front length less than or equal to that of any other invasion for a given constraint network is called

```
program OPTIMAL_INVASION;

procedure TRY(U'; V'; F);
  begin
   T':={t_i | (t_i,R_i) ∈ V'};
   for all Ψ(Ψ⊆U')&( ∃t(∈T-T') s.t. Ψ≤s(t) )
                   &(s(V-(V'∪(∪_{u∈Ψ} τ_u)))=U'-Ψ)                ①

                   &(Fr(<V'∪(∪_{u∈Ψ} τ_u)>_G)<Min_front)  ②  do
     begin
       push Ψ to Dummy_solution;
       if U'-Ψ=φ then
         begin
          Solution:=Dummy_solution;
          Min_front:=F;
         end
        else
          TRY(U'-Ψ; V'∪(∪_{u∈Ψ} τ_u); max{Fr(<V'∪(∪_{u∈Ψ} τ_u)>_G),F});
     end;
  end;{TRY}
begin
 Min_front:=∞;
 TRY(U; φ; 0);
end.{main}
```

Fig. 6 An algorithm for finding an optimal invasion.

optimal. By using Theorem 3 and Theorem 4, we give an algorithm that finds an optimal invasion.

****
* Fig. 6 *
****

The algorithm is shown in Fig. 6, where Solution is a stack which stores unit identifiers. From the bottom to the top, the entries in the stack express the sequence in which the units become redundant. An entry containing two or more unit identifiers indicates that those units become redundant together at some merging-point. In the procedure, the requirement for the Ψ, which is currently being considered as a candidate set of units that may become redundant all at once, is stated at (1) and (2). (1) denotes the characteristics of Theorem 4, explicitly. And (2) checks if the front length for Ψ is less than the front length, Min_front, of the invasion in Solution stack, which is the best one found so far. If that check fails, the procedure rejects pushing down the current Ψ to Dummy_solution as a component of a better invasion. Dummy_solution is also another stack reserving a different possible invasion better than that in Solution. A concrete optimal invasion is derived easily from the result in Solution stack by using Theorem 3.

4. Conclusion

A consistent labeling algorithm, whose basic idea is

breaking the given problem down into several smaller subproblems, has been proposed. It was shown that the concept of an invasion, which gives the order in which the the variables, i.e. the units, of the problem are to be eliminated, plays an major role because it has a large affect on the efficiency of the method.

An algorithm for finding an optimal invasion to reduce the upper bound order of time and space requirements for a given CLP has also been described.

References

[1] Haralick,R.M. and Shapiro,L.G.: The Consistent Labeling Problem, Part I, IEEE Tr. PAMI, 1, 2(1979),173-184.

[2] Seidel,R.: A New Method for Solving Constraint Satisfaction Problems, Proc. IJCAI, 7th(1981),338-342.

[3] Freuder,E.C.: A Sufficient Condition for Backtrack-Free Search, J.ACM, 29, 1(1982),24-32.

[4] Behzad,M., Chartrand,G., and Lesniak-Foster,L.: Graphs and Digraphs, Wadsworth, Inc., Belmont, Calif.(1979).

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-86-55

TITLE

Consistent Labeling Algorithm Using the Dynamic Programming Concept

AUTHOR(S)

Seiichi Nishihara
Tsunemichi Shiozawa
Katsuo Ikeda

REPORT DATE	NUMBER OF PAGES
February 18, 1986	21
MAIN CATEGORY: Problem Solving and Search	CR CATEGORIES: F.2.2, G.2.2, I.2.8

KEY WORDS

consistent labeling, constraint propagation, algorithm, dynamic programming, combinatorial algorithm, graph, constraint network, depth first, breadth first

ABSTRACT

Being given an object consisting of many subparts and their locally legal interpretations, problems of finding totally consistent interpretations are found in many areas, examples of which are image analysis and artificial intelligence. Search problems of this kind are called consistent labeling problems (CLPs). There are two well known principal strategies for the CLP: the depth-first approach typified by backtracking and the breadth-first approach typified by constraint propagation. This paper proposes another approach that makes use of the dynamic programming concept, whose basic idea is reducing the given problem into several smaller subproblems and eliminating variables one by one.

SUPPLEMENTARY NOTES