An Approach to Multiply Segmented Constraint Satisfaction Problems*

Randall A. Helzerman and Mary P. Harper
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907
{helz, harper}@ecn.purdue.edu

Abstract

This paper describes an extension to the constraint satisfaction problem (CSP) approach called MUSE CSP (*MU*ltiply *SE*gmented *C*onstraint *S*atisfaction *P*roblem). This extension is especially useful for those problems which segment into multiple sets of partially shared variables. Such problems arise naturally in signal processing applications including computer vision, speech processing, and handwriting recognition. For these applications, it is often difficult to segment the data in only one way given the low-level information utilized by the segmentation algorithms. MUSE CSP can be used to efficiently represent several similar instances of the constraint satisfaction problem simultaneously. If multiple instances of a CSP have some common variables which have the same domains and compatible constraints, then they can be combined into a single instance of a MUSE CSP, reducing the work required to enforce node and arc consistency.

Introduction

Constraint satisfaction provides a convenient way to represent certain types of problems. In general, these are problems which can be solved by assigning mutually compatible values to a fixed number of variables under a set of constraints. This approach has been used in a variety of disciplines including machine vision, belief maintenance, temporal reasoning, graph theory, circuit design, and diagnostic reasoning (Kumar 1992). A classic example of a CSP is the map coloring problem (e.g., Figure 1), where a color must be assigned to each country such that no two neighboring countries have the same color. A variable represents a country's color, and a constraint arc between two variables indicates that the two joined countries are adjacent and should not be assigned the same color. Formally, a CSP (Mackworth 1977; Mohr & Henderson 1986) is defined in Definition 1.

Definition 1 (Constraint Satisfaction Problem)
$N = \{i, j, \ldots\}$ *is the set of* nodes, *with* $|N| = n$,
$L = \{a, b, \ldots\}$ *is the set of* labels, *with* $|L| = l$,
$L_i = \{a | a \in L$ *and* (i, a) *is admissible*$\}$,
$R1$ *is a* unary relation, (i, a) *is admissible if* $R1(i, a)$,
$R2$ *is a* binary relation, (i, a)-(j, b) *is admissible if* $R2(i, a, j, b)$.

*This work is supported in part by Purdue Research Foundation, NSF grant number IRI-9011179, and NSF Parallel Infrastructure Grant CDA-9015696.

A CSP network contains all n-tuples in L^n which satisfy $R1$ and $R2$. Since some of the values associated with a variable may be incompatible with values assigned to other variables, it is desirable to eliminate as many of these values as possible by enforcing local consistency conditions (such as arc consistency) before a globally consistent solution is extracted (Dechter 1992). Node and arc consistency are defined in Definitions 2 and 3 respectively. Node consistency is easily enforced by the operation $L_i = L_i \cap \{x | R1(i, x)\}$. Enforcing arc consistency is more complicated, but Mohr and Henderson (Mohr & Henderson 1986) have designed an optimal algorithm (AC-4), which runs in $O(yl^2)$ time (where y is the number of pairs of nodes for which $R2$ is not the TRUE relation).

Definition 2 (Node Consistency) *An instance of CSP is said to be* node consistent *if and only if each variable's domain contains only labels which do not violate the unary relation R1, i.e.:* $\forall i \in N : \forall a \in L_i : R1(i, a)$

Definition 3 (Arc Consistency) *An instance of CSP is said to be* arc consistent *if and only if for every pair of nodes i and j, each element of* L_i *(the domain of i) has at least one element of* L_j *for which they both satisfy the binary relation R2, i.e.:* $\forall i, j \in N : \forall a \in L_i : \exists b \in L_j : R2(i, a, j, b)$

There are many types of problems which can be solved by using this approach in a more or less direct fashion. There are also problems which might benefit from the CSP approach, but which are difficult to segment into a single set of variables. This is the class of problems our paper addresses. For example, suppose the map represented in Figure 1 were scanned by a noisy computer vision system, with a resulting uncertainty as to whether the line between regions 1 and 2 is really a border or an artifact of the noise. This situation would yield two CSP problems as depicted in Figure 2. A brute-force approach would be to solve both of the problems, which would be reasonable for scenes containing few ambiguous borders. However, as the number of ambiguous borders increases, the number of CSP networks would grow in a combinatorially explosive fashion. In the case of ambiguous segmentation, it might often be more efficient to merge the constraint networks into a single network which would compactly represent all of them simultaneously, as shown in Figure 3. In this paper, we develop an extension to CSP called MUSE CSP (*MU*ltiply *SE*gmented *C*onstraint *S*atisfaction *P*roblem) to represent multiple instances

of CSP problems.

The initial motivation for extending CSP came from work in spoken language parsing (Zoltowski *et al.* 1992; Harper *et al.* 1992; Harper & Helzerman 1993). The output of a hidden-Markov-model-based speech recognizer is often a list of the most likely sentence hypotheses (i.e., an N-best list) where parsing can be used to rule out the ungrammatical sentence hypotheses. Maruyama (Maruyama 1990a; 1990c; 1990b) has shown that parsing can be cast as a CSP with a finite domain, so constraints can be used to rule out syntactically incorrect sentence hypotheses. However, individually processing each sentence hypothesis provided by a speech recognizer is inefficient since many sentence hypotheses are generated with a high degree of similarity. An alternative representation for a list of similar sentence hypotheses is a word graph or lattice of word candidates which contains information on the approximate beginning and end point of each word. Word graphs are typically more compact and more expressive than N-best sentence lists. In an experiment (Zoltowski *et al.* 1992), word graphs were constructed from three different lists of sentence hypotheses. The word graphs provided an 83% reduction in storage, and in all cases, they encoded more possible sentence hypotheses than were in the original list of hypotheses. Figure 4 depicts a word graph containing eight sentence hypotheses which was constructed from two sentence hypotheses: *Its hard to recognizes speech* and *It's hard to wreck a nice beach.* By structuring the spoken language parsing problem as a MUSE CSP problem, the constraints used to parse individual sentences would be applied to a word graph of sentence hypotheses, eliminating from further consideration all those hypotheses that are ungrammatical. The goal in this case is to utilize constraints to eliminate as many ungrammatical hypotheses as possible, and then to select the best remaining sentence hypothesis (given the word probabilities given by the recognizer).

From CSP to MUSE CSP

If there are multiple, similar instances of a CSP which need to be solved, then separately enforcing node and arc consistency on each instance can often result in much duplicated work. To avoid this duplication, we have combined the multiple instances of CSP into a

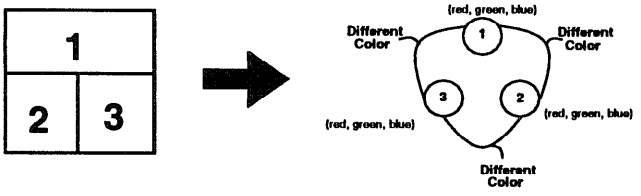


Figure 1: The map coloring problem as an example of CSP. When using a CSP approach, the variables are depicted as circles, where each circle is associated with a finite set of possible values, and the constraints imposed on the variables are depicted using arcs. An arc looping from a circle to itself represents a unary constraint (a relation involving a single variable), and an arc between two circles represents a binary constraint (a relation on two variables).

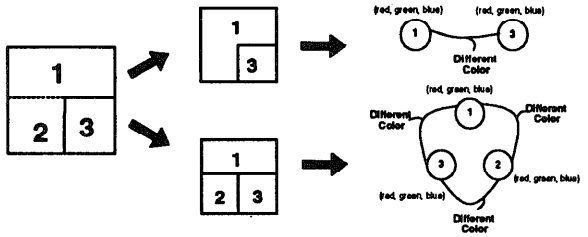


Figure 2: An ambiguous map yields two CSP problems.

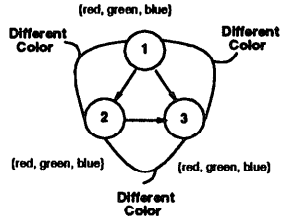


Figure 3: The two CSP problems of figure 2 are captured by a single instance of MUSE CSP. The directed edges form a DAG such that the paths through the DAG correpond to instances of combined CSPs.

shared constraint network and revised the node and arc consistency algorithms to support this representation.

Formally, we define MUSE CSP as follows:

Definition 4 (MUSE CSP)
$N = \{i, j, \ldots\}$ *is the set of* nodes, *with* $|N| = n$,
$\Sigma \subseteq 2^N$ *is a set of* segments *with* $|\Sigma| = s$,
$L = \{a, b, \ldots\}$ *is the set of* labels, *with* $|L| = l$,
$L_i = \{a | a \in L$ *and* (i, a) *is admissible in at least one segment*$\}$,
R1 is a unary relation, (i, a) *is admissible if* $R1(i, a)$,
R2 is a binary relation, (i, a)-(j, b) *is admissible if* $R2(i, a, j, b)$.

The segments in Σ are the different sets of nodes representing instances of CSP which are combined to form a MUSE CSP. We also define $L_{(i,\sigma)}$ to be the set of all labels $a \in L_i$ that are admissible for $\sigma \in \Sigma$.

Because there can be an exponential number of segments in the set Σ, it is important to define methods for combining instances of CSP into a single, compact MUSE CSP. To create a MUSE CSP, we must be able to determine when variables across several instances of CSP can be combined into a single shared variable, and we must also be able to determine which subsets of variables in the MUSE CSP correspond to individual CSPs. A word graph of sentence hypotheses is an

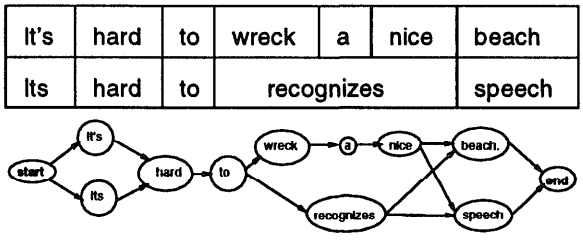


Figure 4: Multiple sentence hypotheses can be parsed simultaneously by propagating constraints over a word graph rather than individual sentences.

excellent representation for a MUSE CSP based constraint parser for spoken sentences. Words that occur in more than one sentence hypothesis over the same time interval are represented using a single variable. The edges between the word nodes, which indicate temporal ordering among the words in the sentence, provide links between words in a sentence hypothesis. A sentence hypothesis, which corresponds to an individual CSP, is simply a path through the word nodes in the word graph going from a start node to an end node.

The concepts used to create a MUSE CSP network for spoken language can be adapted to other CSP problems. In particular, it is desirable to represent a MUSE CSP as a directed acyclic graph (DAG) where the paths through the DAG correspond to instances of CSP problems. In addition, CSP networks should be combined only if they satisfy the conditions below:

Definition 5 (MUSE CSP Combinability) *p instances of CSP $C_1, \ldots, C_p$ are said to be* MUSE Combinable *iff the following conditions hold:*

1. *If* $\{\sigma_1, \sigma_2, \ldots, \sigma_q\} \subseteq \{N_1, \ldots, N_p\} \wedge (i \in \sigma_1 \wedge i \in \sigma_2 \wedge \ldots \wedge i \in \sigma_q) \wedge (a \in L_{(i,\sigma_1)} \wedge a \in L_{(i,\sigma_2)} \wedge \ldots \wedge a \in L_{(i,\sigma_q)})$, *then* $R1_{\sigma_1}(i,a) = R1_{\sigma_2}(i,a) = \ldots = R1_{\sigma_q}(i,a)$.
2. *If* $\{\sigma_1, \sigma_2, \ldots, \sigma_q\} \subseteq \{N_1, \ldots, N_p\} \wedge (i,j \in \sigma_1 \wedge i,j \in \sigma_2 \wedge \ldots \wedge i,j \in \sigma_q) \wedge (a \in L_{(i,\sigma_1)} \wedge a \in L_{(i,\sigma_2)} \wedge \ldots \wedge a \in L_{(i,\sigma_q)}) \wedge (b \in L_{(j,\sigma_1)} \wedge b \in L_{(j,\sigma_2)} \wedge \ldots \wedge b \in L_{(j,\sigma_q)})$, *then* $R2_{\sigma_1}(i,a,j,b) = R2_{\sigma_2}(i,a,j,b) = \ldots = R2_{\sigma_q}(i,a,j,b)$.

These conditions are not overly restrictive, requiring only that the labels for each variable i must be consistently admissible or inadmissible for all instances of CSP which are combined. These conditions do not uniquely determine which variables should be shared across CSP instances for a particular problem type. We define an operator $\oplus$ which combines instances of CSP into an instance of MUSE CSP in Definition 6.

Definition 6 *($\oplus$, the MUSE CSP Combining Operator)* If $C_1, \ldots, C_p$ are MUSE combinable instances of CSP, then $C = C_1 \oplus \ldots \oplus C_p$ will be an instance of MUSE CSP such that:

$$N = N_1 \cup \ldots \cup N_p$$
$$\Sigma = \{N_1, \ldots, N_p\}$$
$$L = L_1 \cup \ldots \cup L_p$$
$$L_i = L_{(i,N_1)} \cup L_{(i,N_2)} \cup \ldots \cup L_{(i,N_p)}$$
$$R1(i,a) = \bigvee_{\sigma=N_1}^{N_p} (R1_\sigma(i,a) \wedge a \in L_{(i,\sigma)})$$
$$R2(i,a,j,b) = \bigvee_{\sigma=N_1}^{N_p} (R2_\sigma(i,a,j,b) \wedge (a \in L_{(i,\sigma)} \wedge b \in L_{(j,\sigma)}))$$

As mentioned above, a DAG is an excellent representation for a MUSE CSP, where its nodes are the elements of N, and its edges are arranged such that every σ in Σ maps onto a path through the DAG. In some applications, such as speech recognition (Zoltowski *et al.* 1992; Harper *et al.* 1992; Harper & Helzerman 1993), the DAG will already be available to us. In applications where the DAG is not available, the user must determine the best way to combine instances of CSP to maximize sharing. We have provided an algorithm (shown in Figure 5) which automatically constructs a single instance of a MUSE

1. Assign each element i of N some number v in the range of 1 to n by using ord(i) = v.
2. **for** each $\sigma \in \Sigma$ **do** {
3. Add to σ two distinguished nodes called **start** and **end** with ord(**start**)=0, ord(**end**)=∞.
4. Sort the elements of σ by their ordinal numbers.
5. **for** $i, j \in \sigma$ such that i's position immediatly preceeds j's position in the sorted σ **do** {
6. Next-edge$_i$:=Next-edge$_i$ $\cup$ $\{(i,j)\}$
7. Prev-edge$_j$:=Prev-edge$_j$ $\cup$ $\{(i,j)\}$ }}

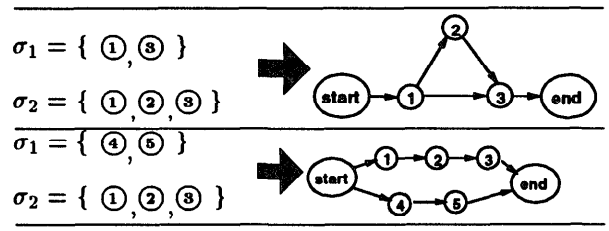


Figure 5: The algorithm to create a DAG to represent the set Σ, and examples of its action.

CSP from multiple CSP instances in $O(sn \log n)$ time, where s is the number of CSP instances to combine, and n is the number of nodes in the MUSE CSP. This algorithm requires the user to assign numbers to variables in the CSPs such that variables that can be shared are assigned the same number. As shown by the examples in Figure 5, for a given set of CSPs, the greater the intersection between the sets of node numbers across CSPs, the more compact the MUSE CSP.

Depending on the application, the solution for a MUSE CSP could range from the set of consistent labels for a single path through the MUSE CSP to all compatible sets of labels for all paths (or CSPs). For our speech processing application, we select the most likely path through the MUSE CSP based on probabilities assigned to word candidates by our speech recognition algorithm. It is desirable to prune the search space before selecting a solution by enforcing local consistency conditions, such as node and arc consistency. However, node and arc consistency must first be extended to MUSE CSP.

Definition 7 (MUSE Node Consistency) *An instance of MUSE CSP is said to be* node consistent *if and only if each variable's domain, L_i, contains only labels which do not violate the unary relation R1, i.e.:* $\forall i \in N : \forall a \in L_i : R1(i,a)$

Definition 8 (MUSE Arc Consistency) *An instance of MUSE CSP is said to be* arc consistent *if and only if for every label a in each domain L_i there is at least one segment σ whose nodes' domains contain at least one label b which satisfy the binary relation R2, i.e.:* $\forall i \in N : \forall a \in L_i : \exists \sigma \in \Sigma : i \in \sigma \wedge \forall j \in \sigma : j \neq i \Rightarrow \exists b \in L_j : R2(i,a,j,b)$

A MUSE CSP is node consistent if all of its segments are node consistent. Unfortunately, arc consistency in a MUSE CSP requires more attention because even though a binary constraint might disallow a label in one segment, it might allow it in another segment. When enforcing arc consistency in a CSP, a label a in L_i can be eliminated from node i whenever any other domain L_j has no labels which together with a satisfy the binary constraints. However, in a MUSE CSP, before a label can be eliminated from a node, it must fail to satisfy the binary constraints in all the segments in

Notation	Meaning
(i,j)	An ordered pair of nodes.
$[(i,j),a]$	An ordered pair of a node pair (i,j) and a label $a \in L_i$.
$M[(i,j),a]$	$M[(i,j),a] = 1$ indicates that the label a is not admissable for (and has already been eliminated from) all segments containing i and j.
E	All node pairs (i,j) such that there exists a segment which contains both i and j.
$S[(i,j),a]$	$[(j,i),b] \in S[(i,j),a]$ means that label a at node i and b at j are simultaneously admissible.
Next-edge$_i$	If a directed edge from i to j exists in E, then (i,j) is a member of this set.
Prev-edge$_i$	If a directed edge from j to i exists in E, then (j,i) is a member of this set.
Counter$[(i,j),a]$	The number of labels in L_j compatible with a in L_i.
Prev-Sup$[(i,j),a]$	$(i,k) \in$ Prev-Sup$[(i,j),a]$ means that a is admissible in every segment containing i,j, and k.
Next-Sup$[(i,j),a]$	$(i,k) \in$ Next-Sup$[(i,j),a]$ means that a is admissible in every segment containing i,j, and k.
Local-Prev-Sup(i,a)	A set of elements (i,j) such that $(j,i) \in$ Prev-edge$_i$ and a is compatible with at least one of j's labels.
Local-Next-Sup(i,a)	A set of elements (i,j) such that $(i,j) \in$ Next-edge$_i$ and a is compatible with at least one of j's labels.
List	A queue of arc support to be deleted.

Figure 6: Data structures and notation for the algorithms.

which it appears. Therefore, the definition of MUSE arc consistency is modified as shown in Definition 8. Notice that Definition 8 reduces to Definition 3 when the number of segments is one. Because MUSE arc consistency must hold for all segments, if a single CSP were selected from the MUSE CSP after MUSE arc consistency is enforced, additional filtering can be required for that single instance.

MUSE CSP Arc Consistency Algorithm

MUSE arc consistency[1] is enforced by removing from the domains those labels in L_i which violate the conditions of Definition 8. MUSE AC-1 builds and maintains several data structures, described in Figure 6, to allow it to efficiently perform this operation. Figure 8 shows the code for initializing the data structures, and Figure 9 contains the algorithm for eliminating inconsistent labels from the domains.

If label a at node i is compatible with label b at node j, then a *supports* b. To keep track of how much support each label a has, the number of labels in L_j which are compatible with a in L_i are counted, and the total is stored in Counter$[(i,j),a]$. The algorithm must also keep track of which labels that label a supports by using $S[(i,j),a]$, which is a set of arc and role value pairs. For example, $S[(i,j),a] = \{[(j,i),b],[(j,i),c]\}$ means that a in L_i supports b and c in L_j. If a is ever invalid for L_i then b and c will loose some of their support. This is accomplished by decrementing Counter$[(j,i),b]$ and Counter$[(j,i),c]$. For regular CSP arc consistency,

[1]We purposely keep our notation and presentation as close as possible to that of Mohr and Henderson in (Mohr & Henderson 1986), in order to aid understanding for those already familiar with the literature on arc consistency.

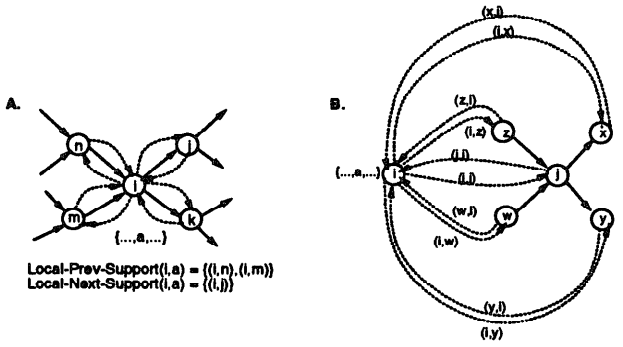


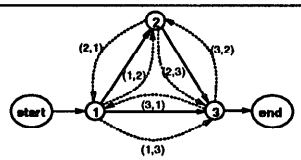
Figure 7: **A**: Local-Prev-Sup and Local-Next-Sup for an example DAG. The sets indicate that the label a is allowed for every segment which contains n, m, and j, but is disallowed for every segment which contains k. **B**: Given that Next-edge$_j$ $= \{(j,x),(j,y)\}$ and $S[(i,x),a] = \emptyset$ and $S[(i,y),a] = \emptyset$, a is inadmissible for every segment containing both i and j.

if Counter$[(i,j),a]$ becomes zero, a would automatically be removed from L_i, because that would mean that a was incompatible with every label for j. However, in MUSE arc consistency, this is not the case, because even though a could not participate in a solution for any of the segments which contain i and j, there could be another segment for which a would be perfectly legal. A label cannot become globally inadmissible until it is incompatible with every segment.

By representing Σ as a DAG, the algorithm is able to use the properties of the DAG to identify local (and hence efficiently computable) conditions under which labels become globally inadmissible. Consider Figure 7A, which shows the nodes which are adjacent to node i in the DAG. Because every segment in the DAG which contains node i is represented as a path through the DAG going through node i, either node j or node k must be in every segment containing i. Hence, if the label a is to remain in L_i, it must be compatible with at least one label in either L_j or L_k. Also, because either n or m must be contained in every segment containing i, if label a is to remain in L_i, it must also be compatible with at least one label in either L_n or L_m.

In order to track this dependency, two sets are maintained for each label a at i, Local-Next-Sup(i,a), and Local-Prev-Sup(i,a). Local-Next-Sup(i,a) is a set of ordered node pairs (i,j) such that $(i,j) \in$ Next-edge$_i$, and there is at least one label $b \in L_j$ which is compatible with a. Local-Prev-Sup(i,a) is a set of ordered pairs (i,j) such that $(j,i) \in$ Prev-edge$_i$, and there is at least one label $b \in L_j$ which is compatible with a. Whenever one of i's adjacent nodes, j, no longer has any labels b in its domain which are compatible with a, then (i,j) should be removed from Local-Prev-Sup(i,a) or Local-Next-Sup(i,a), depending on whether the edge is from j to i or from i to j, respectively. If either Local-Prev-Sup(i,a) or Local-Next-Sup(i,a) becomes the empty set, then a is no longer a part of any solution, and may be eliminated from L_i. In Figure 7A, the role value a is admissible for the segment containing i and j, but not for the

```
1. List:=φ; E := {(i,j)|∃σ ∈ Σ : i,j ∈ σ ∧ i ≠ j ∧ i,j ∈ N};
2. for (i,j) ∈ E do
3.   for a ∈ L_i do {
4.     M[(i,j),a] := 0; Counter[(i,j),a] := 0; S[(i,j),a] := φ;
5.     Prev-Sup[(i,j),a] := φ; Next-Sup[(i,j),a] := φ;
6.     Local-Prev-Sup(i,a) := φ; Local-Next-Sup(i,a) := φ };
7. for (i,j) ∈ E do
8.   for a ∈ L_i do {
9.     Total:=0;
10.    for b ∈ L_j do
11.      if R2(i,a,j,b) then {
12.        Total:=Total+1;
13.        S[(j,i),b] := S[(j,i),b] ∪ {[(i,j),a]} };
14.    if Total=0 then {
15.      M[(i,j),a] := 1; List:=List ∪{[(i,j),a]} };
16.    else Counter[(i,j),a]:=Total;
17.    Prev-Sup[(i,j),a] :=
           {(i,x)|(i,x) ∈ E ∧ (x,j) ∈ Prev-edge_j}
           ∪{(i,j)|(i,j) ∈ Prev-edge_j}
           ∪{(i,start)|(start,j) ∈ Prev-edge_j};
18.    Next-Sup[(i,j),a] :=
           {(i,x)|(i,x) ∈ E ∧ (j,x) ∈ Next-edge_j}
           ∪{(i,j)|(j,i) ∈ Next-edge_j}
           ∪{(i,end)|(j,end) ∈ Next-edge_j};
19.    if (i,j) ∈ Next-edge_i then
         Local-Next-Sup(i,a):=Local-Next-Sup(i,a) ∪ {(i,j)};
20.    if (j,i) ∈ Prev-edge_i then
         Local-Prev-Sup(i,a):=Local-Prev-Sup(i,a) ∪ {(i,j)} }
```



Prev-Sup[(1,2),a] = {(1,2)}	Next-Sup[(1,2),a] = {(1,3)}
Prev-Sup[(1,3),a] = {(1,2),(1,3)}	Next-Sup[(1,3),a] = {(1,end)}
Prev-Sup[(2,1),a] = {(2,start)}	Next-Sup[(2,1),a] = {(2,1),(2,3)}
Prev-Sup[(3,1),a] = {(3,start)}	Next-Sup[(3,1),a] = {(3,1),(3,2)}
Prev-Sup[(2,3),a] = {(2,3),(2,1)}	Next-Sup[(2,3),a] = {(2,end)}
Prev-Sup[(3,2),a] = {(3,1)}	Next-Sup[(3,2),a] = {(3,2)}
Local-Prev-Sup(1,a) = {(1,start)}	Local-Next-Sup(1,a) = {(1,2),(1,3}
Local-Prev-Sup(2,a) = {(2,1)}	Local-Next-Sup(2,a) = {(2,3)}
Local-Prev-Sup(3,a) = {(3,1),(3,2)}	Local-Next-Sup(3,a) = {(3,end)}

Figure 8: Algorithm for initializing the MUSE CSP data structures along with a simple example. The dotted lines are members of the set E.

segment containing i and k. If because of constraints, the labels in j become inconsistent with a on i, (i,j) would be eliminated from Local-Next-Sup(a,i), leaving an empty set. In that case, a would no longer be supported by any segment.

The algorithm can utilize similar conditions for nodes which may not be directly connected to i by Next-edge$_i$ or Prev-edge$_i$. Consider Figure 7B. Suppose that the label a at node i is compatible with a label in L_j, but it is incompatible the labels in L_x and L_y, then it is reasonable to eliminate a for all segments containing both i and j, because those segments would have to include either node x or y. To determine whether a role value is admissible for a set of segments containing i and j, we calculate Prev-Sup$[(i,j),a]$ and Next-Sup$[(i,j),a]$ sets. Next-Sup$[(i,j),a]$ includes all (i,k) arcs which support a in i given that there is a directed edge between j and k and (i,j) supports a. Prev-Sup$[(i,j),a]$ includes all (i,k) arcs which support a in i given that there is a directed edge between k and j and (i,j) supports a. Note that Prev-Sup$[(i,j),a]$ will contain an ordered pair (i,j) if $(i,j) \in$ Prev-edge$_j$, and Next-Sup$[(i,j),a]$ will contain an ordered pair (i,j) if $(j,i) \in$ Next-edge$_j$. These elements are included because the edge between nodes i and j is sufficient to allow j's labels to support a in the segment containing i and j. Dummy ordered pairs are also created to handle cases where a node is at the beginning or end of a network: when $(\mathbf{start}, j) \in$ Prev-edge$_j$, $(i, \mathbf{start})$ is added to Prev-Sup$[(i,j),a]$, and when $(j, \mathbf{end}) \in$ Next-edge$_j$, $(i, \mathbf{end})$ is added to Next-Sup$[(i,j),a]$. Figure 8 shows the support sets that the initialization algorithm creates for the label a in the simple example DAG.

To illustrate how these data structures are used in MUSE AC-1 (see Figure 9), consider what happens if initially $[(1,3),a] \in$ *List* for the MUSE CSP in Figure 8. First, it is necessary to remove $[(1,3),a]$'s support from all S$[(3,1),x]$ such that $[(3,1),x] \in$ S$[(1,3),a]$ by decrementing for each x, Counter$[(3,1),x]$ by one. If the counter for any $[(3,1),x]$ becomes 0, and the value has not already been placed on the *List*, then it is added for future processing. Once this is done, it is necessary to remove $[(1,3),a]$'s influence on the DAG. To handle this, we examine the two sets Prev-Sup$[(1,3),a] = \{(1,2),(1,3)\}$ and Next-Sup$[(1,3),a] = \{(1,\mathbf{end})\}$. Note that the value $(1,\mathbf{end})$ in Next-Sup$[(1,3),a]$ and the value $(1,3)$ in Prev-Sup$[(1,3),a]$, once eliminated from those sets, require no further action because they are dummy values. However, the value $(1,2)$ in Prev-Sup$[(1,3),a]$ indicates that $(1,3)$ is a member of Next-Sup$[(1,2),a]$, and since a is not admissible for $(1,3)$, $(1,3)$ should be removed from Next-Sup$[(1,2),a]$, leaving an empty set. Note that because Next-Sup$[(1,2),a]$ is empty and assuming that M$[(1,2),a] = 0$, $[(1,2),a]$ is added to *List* for further processing. Next, $(1,3)$ is removed from Local-Next-Sup$(1,a)$, but that set is non-empty. During the next iteration of the while loop, $[(1,2),a]$ is popped from *List*. When Prev-Sup$[(1,2),a]$ and Next-Sup$[(1,2),a]$ are processed, Next-Sup$[(1,2),a] = \emptyset$ and Prev-Sup$[(1,2),a]$ contains only a dummy, which is removed. When $(1,2)$ is removed from Local-Next-Sup$(1,a)$, the set becomes empty, so a is no longer compatible with any segment containing 1 and can be eliminated from further consideration as a possible label for 1.

In contrast, consider what happens if initially $[(1,2),a] \in$ *List* for the MUSE CSP in Figure 8. In this case, Prev-Sup$[(1,2),a]$ contains $(1,2)$ which requires no additional work; whereas, Next-Sup$[(1,2),a]$ contains $(1,3)$, indicating that $(1,2)$ must be removed from Prev-Sup$[(1,3),a]$'s set. After the removal, Prev-Sup$[(1,3),a]$ is non-empty, so the segment containing nodes 1 and 3 still supports the label a on 1. The reason that these two cases provide different results is that nodes 1 and 3 are in every segment; whereas, nodes 1 and 2 are only in one of them.

Running Time and Correctness of MUSE AC-1

The running time of the routine to initialize the MUSE CSP data structures (in Figure 8) is $O(n^2l^2+n^3l)$, and the running time for the algorithm which prunes labels that are not arc consistent (in Figure 9) also operates in $O(n^2l^2 + n^3l)$ time, where n is the number of nodes

```
1. while List ≠ φ do {
2.   choose [(j,i),b] from List and remove it from List;
3.   for [(i,j),a] ∈ S[(j,i),b] do {
4.     Counter[(i,j),a]:=Counter[(i,j),a] − 1;
5.     if Counter[(i,j),a] = 0 ∧ M[(i,j),a] = 0 then {
6.       List:=List ∪{[(i,j),a]};
7.       M[(i,j),a] := 1 } };
8.   for (j,x) ∈ Next-Sup[(j,i),b] do {
9.     Prev-Sup[(j,x),b]:=Prev-Sup[(j,x),b] − {(j,i)};
10.    if Prev-Sup[(j,x),b] = φ ∧ M[(j,x),b] = 0 then {
11.      List:=List ∪{[(j,x),b]};
12.      M[(j,x),b] := 1 } };
13.  for (j,x) ∈ Prev-Sup[(j,i),b] do {
14.    Next-Sup[(j,x),b]:=Next-Sup[(j,x),b] − {(j,i)};
15.    if Next-Sup[(j,x),b] = φ ∧ M[(j,x),b] = 0 then {
16.      List:=List ∪{[(j,x),b]};
17.      M[(j,x),b] := 1 } };
18.  if (j,i) ∈ Next-edge_j then
19.    Local-Next-Sup(j,b):= Local-Next-Sup(j,b) − {(j,i)};
20.  if Local-Next-Sup(j,b) = φ then {
21.    L_j := L_j − {b}
22.    for (j,x) ∈ Local-Prev-Sup(j,b) do
23.      if M[(j,x),b] = 0 then {
24.        List:=List ∪{[(j,x),b]}; M[(j,x),b] := 1 } };
25.  if (i,j) ∈ Prev-edge_j then
26.    Local-Prev-Sup(j,b):= Local-Prev-Sup(j,b) − {(j,i)};
27.  if Local-Prev-Sup(j,b) = φ then {
28.    L_j := L_j − {b}
29.    for (j,x) ∈ Local-Next-Sup(j,b) do
30.      if M[(j,x),b] = 0 then {
31.        List:=List ∪{[(j,x),b]}; M[(j,x),b] := 1 } } } ;
```

Figure 9: Algorithm to enforce MUSE CSP arc consistency.

in a MUSE CSP and l is the number of labels. By comparison, the running time for CSP arc consistency is (n^2l^2), assuming that there are n^2 constraint arcs. Note that for applications where $l = n$, the running times of the algorithms are the same (this is true for parsing spoken language with a MUSE CSP). Also, if the Σ is representable as a planar DAG (in terms of Prev-edge and Next-Edge, not E), then the running time of the algorithms is the same because the average number of values in Prev-Sup and Next-Sup would be a constant. In the general case, the increase in the running time for arc consistency of a MUSE CSP is reasonable considering that it is possible to combine a large number of CSP instances (possibly exponential) into a compact graph with a small number of nodes.

Next we prove the correctness of MUSE AC-4. A role value is eliminated from a domain by MUSE AC-4 only if its Local-Prev-Sup or its Local-Next-Sup becomes empty. Therefore, we must show that a role value's local support sets become empty if and only if that role value cannot participate in a MUSE arc consistent solution. This is proven for Local-Next-Sup (Local-Prev-Sup follows by symmetry). Observe that if $a \in L_i$, and it is incompatible with all of the nodes which immediately follow L_i in the DAG, then it cannot participate in a MUSE arc consistent solution. In line 19 in figure 9, (i,j) is removed from Local-Next-Sup(i,a) set only if $[(i,j),a]$ has been popped off *List*. Therefore, we show that $[(i,j),a]$ is put on *List*, only if $a \in L_i$ is incompatible with every segment which contains i and j by induction on the number of iterations of the while loop.

For the base case, the initialization routine only puts $[(i,j),a]$ on *List* if $a \in L_i$ is incompatible with every label in L_j (line 15 of Figure 8). Therefore, $a \in L_i$ is in no solution for any segments which contain i and j. Assume the condition holds for the first k iterations of the while loop in Figure 9, then during the (k+1)th iteration, new tuples of the form $[(i,j),a]$ are put on the list by line 6 (in which case a is no longer compatible with any labels in L_j), line 11 (in which case Prev-Sup$([(i,j),a]) = \phi$), line 16 (in which case Next-Sup$([(i,j),a]) = \phi$), or line 24 (there is no longer any Next-Sup for a). In any of these cases, $a \in L_i$ is incompatible with every segment which contains i and j. We can therefore conclude that this is true for all iterations of the while loop.

In conclusion, MUSE CSP can be used to efficiently represent several similar instances of the constraint satisfaction problem simultaneously. If multiple instances of a CSP have some common variables with the same domains and compatible constraints, then they can be combined into a single instance of a MUSE CSP, and much of the work required to enforce node and arc consistency need not be duplicated across the instances. For our work in speech processing, the MUSE arc consistency algorithm was very effective at pruning the incompatible labels for the individual CSPs represented in the composite structure. Very little additional work is typically needed to enforce arc consistency on a CSP represented by the best path through the network.

References

Dechter, R. 1992. From local to global consistency. *Artificial Intelligence* 55:87–107.

Harper, M. P., and Helzerman, R. A. 1993. PARSEC: A constraint-based parser for spoken language parsing. Technical Report EE-93-28, Purdue University, School of Electrical Engineering, West Lafayette, IN.

Harper, M.; Jamieson, L.; Zoltowski, C.; and Helzerman, R. 1992. Semantics and constraint parsing of word graphs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, II–63–II–66.

Kumar, V. 1992. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine* 13(1):32–44.

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

Maruyama, H. 1990a. Constraint dependency grammar. Technical Report #RT0044, IBM, Tokyo, Japan.

Maruyama, H. 1990b. Constraint dependency grammar and its weak generative capacity. *Computer Software.*

Maruyama, H. 1990c. Structural disambiguation with constraint propagation. In *The Proceedings of the Annual Meeting of ACL.*

Mohr, R., and Henderson, T. C. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Zoltowski, C.; Harper, M.; Jamieson, L.; and Helzerman, R. 1992. PARSEC: A constraint-based framework for spoken language understanding. In *Proceedings of the International Conference on Spoken Language Understanding.*