Tree Cover Search Algorithm for Example-Based Translation

Hiroshi Maruyama Hideo Watanabe
IBM Research, Tokyo Research Laboratory
5-19, Sanbancho, Chiyoda-ku, Tokyo 102, JAPAN
{maruyama,watanabe}@trl.vnet.ibm.com

Abstract

Efficient tree-cover search is essential for the realization of practical example-based translation systems. This paper presents an efficient algorithm for finding the optimum combination of translation examples that covers the whole input tree based on the dynamic-programming technique.

1 Introduction

Example-based approach is a new paradigm that has been recently studied extensively. The basic idea of it is to use an example similar to a phenomenon in question as a knowledge instead of heuristics. This EBT approach is especially promising for use in transfer modules, because these involve a problem dealing with idiosyncracy for which the EBT approach is more suitable than rules written according to human's linguistic intuition.

Some research has already been done on the EBT approach [1, 2, 3, 5, 6, 7]. Essentially the systems that have been developed collect a large number of translation pairs and translate a given input sentence on the pattern of similar translation examples. To implement such a system, the following issues should be resolved:

- Similarity Calculation - To calculate the similarity between an input and a source part of a translation example.
- Example Selection - To select the most appropriate set of translation examples.
- Target Construction - To construct a target sentence by combining the target parts of the chosen examples.

One of the major problems in the EBT approach is how to select a set of examples. Since it is rare for an example to have an one-to-one matching with an input sentence, we must select a set of example fragments that completely covers the input sentence. For instance, assume that the input sentence is represented as (a) in Figure 1, and that (b), (c), (d), and (e) are translation examples. The problem is to find the the most appropriate combination of the examples that completely covers the input structure (a). In this case, {(b),(e)} and {(c),(d)} are the candidates of covers. Suppose that matching cost of (b) is 2, (c) is 1, (d) is 5, and (e) is 1, then matching cost of {(b),(e)} is smaller than that of {(c),(d)}. Therefore, {(b),(e)} should be selected as the most appropriate cover of (a). Since there are combinatorial number of such combinations in general, selecting the most appropriate example set is by no means trivial, and this can be a bottleneck of practical EBT systems. As mentioned by Sato [4], few studies have been made of this problem. This paper proposes an efficient method based on dynamic programming, for selecting the most appropriate set of translation examples.

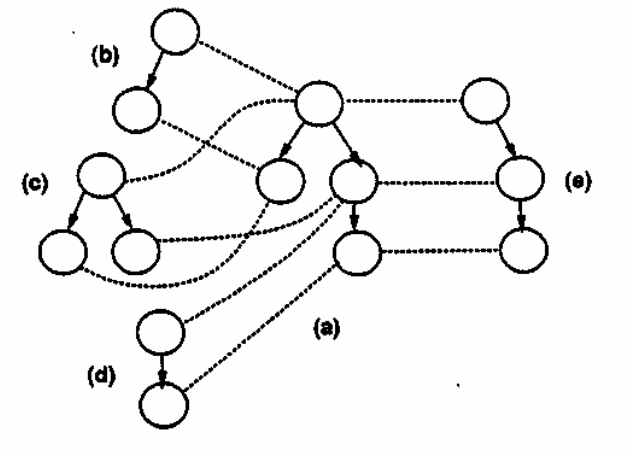

Figure 1: Selecting the most appropriate set of rules.

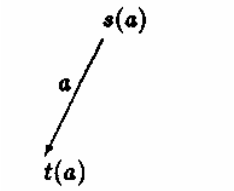

Figure 2: arc *a*, source node *s(a),* destination node *t(a)*

2 Tree-cover search algorithm

2.1 Problem definition

We assume that the linguistic structure of an input sentence is represented by a tree. In this section, an arc of a tree is represented by an arrow, as shown in Figure 2. Let *a* be an arc. We define *s(a)* as the source node of *a*, and *t(a)* as the destination node of *a*. Each node is associated with a set of linguistic properties such as a part of speech and semantic features, but since the details of these properties are irrelevant to the algorithm that we present, they are omitted in this section (practical examples with linguistic properties are given in Section 3).

The linguistic structure of the source language part in a translation example is also represented by a tree. We call such trees $P_1, P_2,...,P_r$ *patterns,* and use the notation $\mathcal{P}$ to represent a set of patterns $\{P_1, P_2, ...,P_r\}$[1].

Definition 1 (Match) *Let n be a node in an input tree T and let* $P \in \mathcal{P}$ *be a pattern.* $\langle n, P, f \rangle$ *is a* match *if and only if f is a one-to-one mapping defined for every arc of P and satisfies the following conditions (see Figure 3). f is called a* matching function *of* $\langle n, P, f \rangle$.

1. Let p be the root node of P. For any arc a in P, if p = s(a) then n = s(f(a)).

[1] In this paper, we use *S,T,...* for trees, *n, m,...* for nodes, *a,b,...* for arcs, and $\mathcal{P}, \mathcal{Q}, \ldots$ for set of trees, unless stated otherwise.

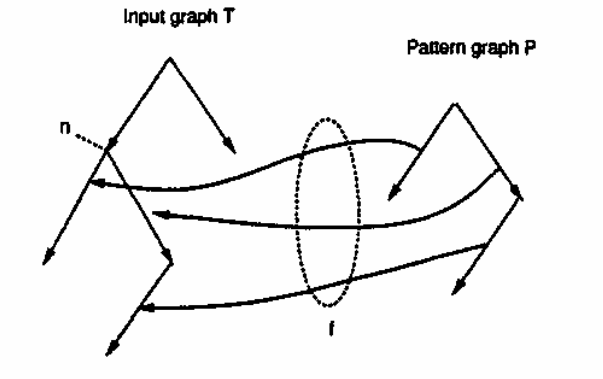

Figure 3: Matching between an input and a pattern

2. *For any pair of arcs a, b in P, if s(a) = s(b) then s(f(a)) = s(f(b)).*
3. *For any pair of arcs a, b in P, if s(a) = t(b) then s(f(a)) = t(f(b)).*

Definition 2 (Covered arc) *An arc b in the input tree is* covered by a match $\langle n, P, f \rangle$ *if and only if there is an arc a in the pattern P such that b = f(a).*

Definition 3 (Cover) *Set C of matches* $\langle n_1,P_1,f_1 \rangle$, $\langle n_2,P_2,f_2 \rangle$, …, $\langle n_k,P_k,f_k \rangle$ *is called a* cover *if and only if*

1. *the arcs covered by C are connected, and*
2. *no arc in the input tree is covered by more than one arc in C .*

Assume that a nonnegative cost function *cost(n, P, f)* is given for match $\langle n, P, f \rangle$. The cost will be calculated according to the linguistic properties attached to nodes and arcs. The value infinity is given to linguistically illegal matches. Using this cost function for a match, we define the cost of a cover by the following sum of matches:

$$cost(C) = \sum_{\langle n, P, f \rangle \in C} cost(n, P, f).$$

Our goal is to find the minimum-cost cover that covers every arc in the input tree.

2.2 Minimum-cost cover

We construct the minimum-cost cover of the whole tree from the minimum-cost covers of subtrees. For this purpose, we define several notions:

Definition 4 (Closed node, Open node) *Let C be a cover and let n be a node in the input tree. n is* closed *if and only if every arc originating at n is covered by C. Node n is* open *if and only if n is not closed (see Figure 4).*

Note that leaf nodes are closed by definition.

Definition 5 (Well-formed cover) *Cover C is* well-formed *for input node n if and only if it satisfies the following three conditions (see Figures 5 and 6):*

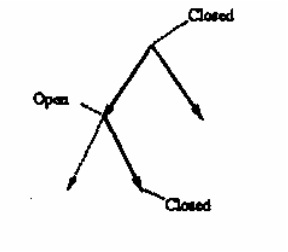

Figure 4: Open nodes and closed nodes (covered arcs are denoted by thick arrows, and uncovered arcs by dotted arrows)

Figure 5: Examples of well-formed covers

1. *If any arcs originate at node n, at least one of them is covered byC.*
2. *No ancestor nodes of n are covered by C.*
3. *For any descendant node x of n, if x is open then there is a path from n to x whose arcs are not covered by C .*

Property 1 *Let C be a well-formed cover that covers every arc originating at the root node n_0 of the input tree T. Then every arc in T is covered by C.*

Proof: Assume that an arc *a* is not covered by *C*. Let *x* be the origin node of *a (x = s(a))*. Since n_0 is closed, $x \neq n_0$. Because *a* is not covered by *C, x* is open, and therefore there is a path from n_0 to *x* whose arcs are not covered by $\subset$. However, since n_0 is closed, no such path exists.

□

This property implies that, to obtain a cover that covers all the arcs in the input tree, it suffices to find a well-formed cover of the root node such that n_0 is closed.

Figure 6: Examples of ill-formed covers

2.3 Constructing a minimum-cost cover

We construct well-formed covers from the leaf nodes to the root node. For a given node *n*, there are two ways of constructing a cover of *n*. One is to fix a match of *n* and then to fill the open nodes of the match with the well-formed covers of these open nodes as explained in the next theorem:

Theorem 1 (Constructing a cover of node n) *Let ⟨n, P, f⟩ be a match and let X be the set of all descendant open nodes of n with regard to ⟨n, P, f⟩. For every member x ∈ X, let C_x be a well-formed cover of x that covers every arc originating at x not covered by ⟨n, P, f⟩ and that does not cover arcs covered by ⟨n, P, f⟩. Then,*

$$C' = \{\langle n, P, f\rangle\} \cup \bigcup_{x \in X} C_x$$

is a well-formed cover of n. In addition,

$$cost(C') = cost(\langle n, P, f\rangle) + \sum_{x \in X} cost(\mathcal{M}_x)$$

Proof: Since no arcs are duplicated, *C′* is a cover. Since all the descendant nodes of *n* are closed, *C′* is well-formed.

□

The other way of constructing a cover of node *n* is to merge two mutually disjoint covers of *n* and to make a larger well-formed cover of *n*.

Theorem 2 (Combining two covers of node n) *Let C_1 and C_2 be mutually disjoint well-formed covers of n (that is, no arcs are covered by both C_1 and C_2). Then $C_1 \cup C_2$ is a well-formed cover of n. In addition,*

$$cost(C_1 \cup C_2) = cost(C_1) + cost(C_2)$$

Proof: Let *y* be an arbitrary descendant node of *n*. If *y* is covered by either C_1 or C_2, *y* is closed. Otherwise, there is a path from *n* to *y* whose arcs are not covered by $C_1 \cup C_2$.

□

In general, there are two or more well-formed covers for one node. Some of them cover exactly the same set of arcs. However we want to store only the one with the lowest cost. For this purpose, we define the *isomorphism* relation between well-formed covers.

Definition 6 (Isomorphic cover) *Let C_1 and C_2 be well-formed covers of node n. C_1 and C_2 are* isomorphic *(written as $C_1 \sim C_2$) if and only if they cover exactly the same set of arcs.*

It is obvious that the relation ~ is an equivalence relation. Therefore, given a node *n*, the set of all well-formed covers of *n* can be divided into equivalent sets according to ~. For example, if the degree of node *n* is two, the well-formed covers of *n* are divided into three equivalence sets D_1, D_2, and D_3 as shown in Figure 7.

For a node with degree *d,* there are 2^d - 1 equivalence sets. Each equivalence set may have many well-formed covers, but we only need the one with the lowest cost. Therefore, an array attached to the node with size 2^d - 1 can hold this information. We have to be careful, however, because there are certain dependencies between the minimum costs of the equivalence sets. For example, the minimum costs of D_1 and D_2 in Figure 7 should be obtained *before* the minimum cost of D_3 is calculated. For this purpose, we number the equivalence sets of node *n* as D_1, D_2,... in ascending order of the number of arcs originating at *n*. Our algorithm first builds covers according to Theorem 1 (cover construction from a match), and then for each D_1, D_2,..., D_{2^d-1}, combines each pair of the minimum cost covers of the equivalence sets using Theorem 2 (cover combination).

For convenience, when D_1 and D_2 are the equivalence sets of C_1 and C_2 , respectively, the equivalence set of $C_1 \cup C_2$ is written as $D_1 \oplus D_2$. Thus, if $D_k = D_i \oplus D_j$, then $k > i$ and $k > j$.

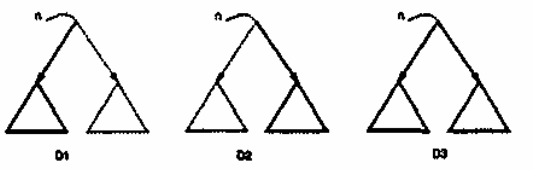

Figure 7: Equivalence sets of well-formed covers of a node with degree=2

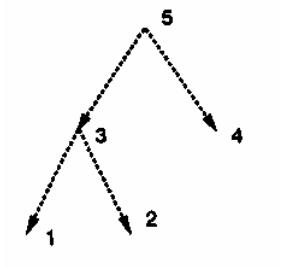

Figure 8: Numbering nodes

2.4 Algorithm

Based on the general strategy of dynamic programming, our algorithm solves subproblems and stores their solutions to avoid duplicate calculation. For each n, the number of equivalence sets is $2^{d(n)} - 1$ ($d(n)$ is the degree of node n). Therefore, we provide the following two arrays for keeping information on the minimum-cost covers of subtrees and their costs:

bestCover[l..($2^{d(n)}$ - 1)][1..N]
bestScore[l..($2^{d(n)}$ - 1)][1..N],

where the capital letter N is the number of nodes in the input tree T. We assume that the nodes of T are numbered from 1 to N in the post-fix order (see Figure 8). Our full algorithm is shown in Figure 9.

The main loop (the one beginning with line (3)) constructs the minimum cost covers from bottom to top. The body of the main loop consists of two sub-loops (the one beginning with line 4 and the one beginning with line 16) corresponding Theorem 1 and Theorem 2, respectively.

2.5 Complexity

The complexity of this algorithm depends on the degree of nodes in the input tree T. However, in our application, the degree of each node is mostly limited to a very small integer. Hence, we can assume that the degree is a constant. For simplicity, we discuss the complexity on the assumption that the degree of each node is at most two (that is, the input tree is a binary tree).

Let N be the number of nodes in the input tree, let p be the number of patterns, and let a be the maximum size of the patterns. The main loop of the algorithm (line 3) is executed N times. The body of the loop is divided into the construction of well-formed covers of node n from the well-formed covers of the descendant nodes, and the merging of the well-formed covers of node n. Let us consider the first of these parts.

The outer loop of this first part is executed as many times as there are patterns p. When the node n and the pattern p are fixed, there may be $O(2^a)$ matches in the worst case. Therefore, the computational complexity of the first part is $O(ap2^a)$.

```
bestCover[•][•] ← {}                                                              (1)
bestScore[•][•] ← ∞                                                               (2)
for n := 1 to N begin                                                             (3)
    for P ∈ 𝒫 begin                                                               (4)
        for every f such that (n, P, f) is a match and cost(n, P, f) < ∞ begin    (5)
            D ← the equivalence set of the arcs covered by (n, P, f)              (6)
            ℳ ← {(n, P, f)}                                                       (7)
            c ← cost(n, P, f)                                                     (8)
            for every descendant open node x of n begin                           (9)
                D' ← the equivalence set of a well-formed cover such that x is closed   (10)
                ℳ ← ℳ ∪ bestCover[D'][x]                                          (11)
                c ← c + bestScore[D'][x]                                          (12)
            end
            if c < bestScore[D][n] then begin                                     (13)
                bestCover[D][n] ← ℳ                                               (14)
                bestScore[D][n] ← c                                               (15)
            end
        end
    end
    for k := 1 to 2^d(n) − 1 begin                                                (16)
        for every i, j such that D_k = D_i ⊕ D_j begin                            (17)
            ℳ ← bestCover[D_i][n] ∪ bestCover[D_j][n]                             (18)
            c ← bestScore[D_i][n] + bestScore[D_j][n]                             (19)
            if c < bestScore[D_k][n] then begin                                   (20)
                bestCover[D_k][n] ← ℳ                                             (21)
                bestScore[D_k][n] ← c                                             (22)
            end
        end
    end
end
```

Figure 9: Algorithm

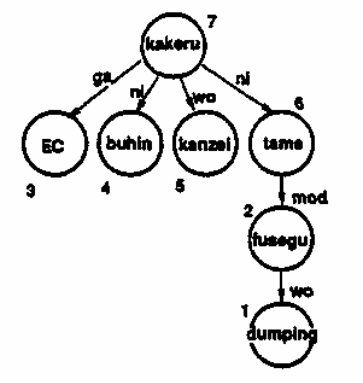

Figure 10: Japanese Dependency Structure

The second half of the body of the outmost loop is executed at most six times (${}_3C_2$) provided that the degree of n is two. Therefore, the overall computational time is bounded by $O(Nap2^a)$. This is linear with respect to the input size N. Note that we intend to find a cover by combining very small fragments of translation patterns, which implies that a is much smaller than the input size N.

3 Example

This section gives a concrete example of how to make a minimum-cost cover by using our algorithm. Let us take the following Japanese sentence as an example:

EC ga danpingu wo fusegu tameni buhinn ni kanzei wo kaketa.
(The EC imposed customs duties on parts in order to prevent dumping. (English))

This Japanese sentence is analyzed and converted into the JDS (Japanese Dependency Structure) shown in Figure 3. We consider the five Japanese-to-English translation patterns shown in Figure 11. In each translation pattern, the left-hand structure is a JDS, the right-hand structure is an EDS (English Dependency Structure), and dotted arrows represent correspondences between the JDS and EDS.

The cost used in the algorithm is a distance calculated on the basis of similarity, but for reasons of space we cannot describe the details of the calculation in this paper. Further, to clarify the coverage of an equivalence set, we denote the set as a sequence of its root node number and the numbers of the nodes covered by the set (for example, D734).

For the sub-tree in Figure 12 whose root is "fusegu," pt2 makes a well-formed cover that has the lowest cost 1.5 (we denote its match pt2:21(1.5)), and it is the minimum-cost pattern in its equivalence set, D21. D21 is the minimum-cost cover for the "fusegu" sub-tree, because the sub-tree has only one equivalence set, D21. Subsequently, for the sub-tree whose root is "tame," combination of pt6:62(2.0), which matches "fusegu tame," and the minimum-cost cover of the "fusegu" sub-tree, D21, gives {pt6:62+D21}(3.5) as the minimum-cost well-formed cover, and this becomes the minimum-cost cover of its equivalence set, D62. D62 is also the minimum-cost cover of the "tame" sub-tree, because this sub-tree has only one equivalence set.

Matches for the sub-tree whose root is "kakeru" are shown in Figure 13. First, we consider the cover equivalence sets D73, D74, D75, and D76, which have only one arc each. As shown in Figure 13, pt l:73, pt3:74, and pt2:75 are the minimum-cost well-formed covers for D73, D74, and D75, respectively. For D76 (the sub-tree "tame ni kakeru"), the following two combinations of covers are possible:

D76: (a1) cost(pt3:76)+cost(D62) = 1.5+3.5 = 5.0
(a2) cost(pt5:72)+cost(D21) = 0.3+1.5 = 1.8

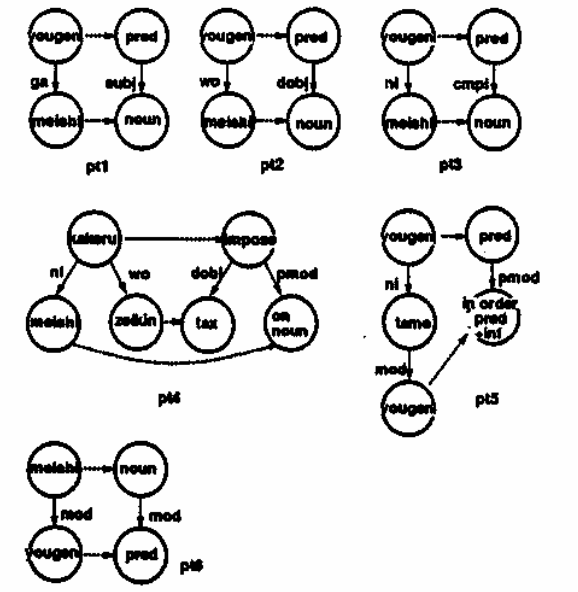

Figure 11: J-to-E Translation Patterns

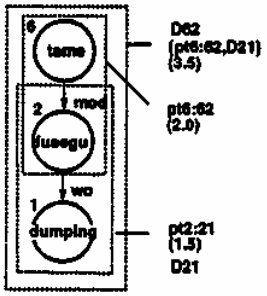

Figure 12: Matches and best-covers for "fusegu" and "tame"

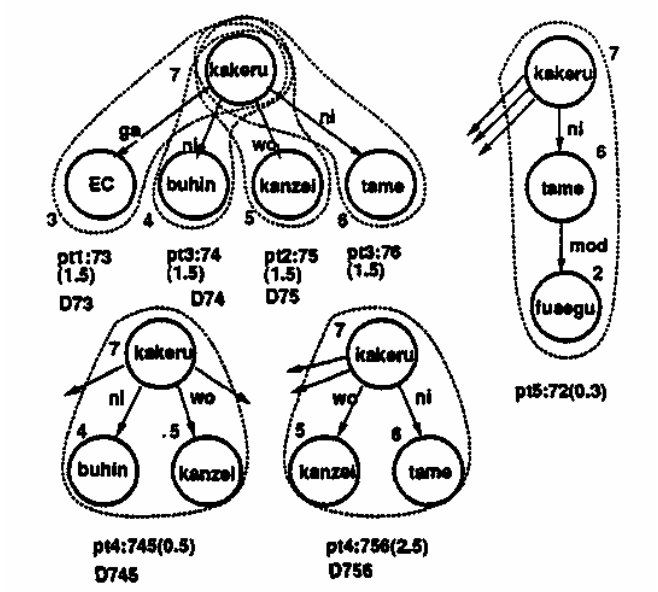

Figure 13: Matches for "kakeru" sub-tree

Hence, (a2) is the minimum-cost well-formed cover of D76.

Next, we consider the cover equivalence sets D734, D735, D736, D745, D746, and D756, which have two arcs each. The following are the well-formed covers for these cover equivalence sets:

D734: (bl) cost(D73)+cost(D74) = 1.5+1.5 = 3.0
D735: (b2) cost(D73)+cost(D75) = 1.5+1.5 = 3.0
D736: (b3) cost(D73)+cost(D76) = 1.5+1.8 = 3.3
D745: (b4) cost(D74)+cost(D75) = 1.5+1.5 = 3.0
(b5) cost(pt4:745) = 0.5
D746: (b6) cost(D74)+cost(D76) = 1.5+1.8 = 3.3
D756: (b7) cost(D75)+cost(D76) = 1.5+1.8 = 3.3
(b8) cost(pt4:756) = 2.5

Hence, (bl) is the minimum-cost well-formed cover for D734, (b2) for D735, (b3) for D736, (b5) for D745, (b6) for D746, and (b8) for D756.

Further, we consider the cover equivalence sets D7345, D7346, and D7456 which have three arcs each. The following are the well-formed covers for these cover equivalence sets:

D7345: (cl) cost(D73)+cost(D745) = 1.5+0.5 = 2.0
(c2) cost(D734)+cost(D75) = 3.0+1.5 = 4.5
D7346: (c3) cost(D73)+cost(D746) = 1.5+3.3 = 4.8
(c4) cost(D734)+cost(D76) = 3.0+1.8 = 4.8
D7356: (c5) cost(D73)+cost(D756) = 1.5+2.5 = 4.0
(c6) cost(D735)+cost(D76) = 3.0+1.8 = 4.8
D7456: (c7) cost(D74)+cost(D756) = 1.5+2.5 = 4.0
(c8) cost(D745)+cost(D76) = 0.5+1.8 = 2.3

Hence, (cl) is the minimum-cost well-formed cover for D7345, (c3) and (c4) for D7346 (they have the same cost 4.8), (c5) for D7356, and (c8) for D7456.

Finally, let us find the minimum-cost cover for the whole tree. The possible combinations of covers are as follows:

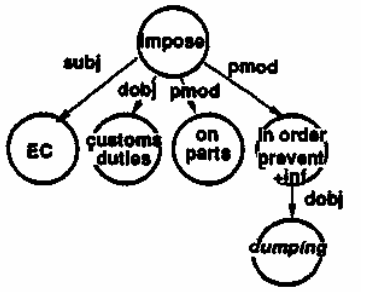

Figure 14: English dependency structure

D73456:

(dl) cost(D73)+cost(D7456) = 1.5+2.3 = 3.8
(d2) cost(D74)+cost(D7356) = 1.5+4.0 = 5.5
(d3) cost(D75)+cost(D7346) = 1.5+4.8 = 6.3
(d4) cost(D76)+coat(D7345) = 1.8+2.0 = 3.8
(d2) cost(D734)+cost(D756) = 3.0+2.5 = 5.5

Although (dl) and (d4) have the same lowest cost, they consist of the same equivalence sets {D73,D745,D76}. Hence, {ptl:73,pt4:745,pt5:72,pt2:21} becomes the minimum-cost cover for the input sentence.

Figure 14 shows a complete EDS constructed from the EDSs of translation patterns in the above minimum-cost cover. Briefly, this target structure is formed in such a way that, in the EDSs of translation patterns, nodes that correspond to the same source node are unified. (See [7] for details.)

The English generator makes a surface English string from the above EDS as follows:

"The EC imposed customs duties on parts
in order to prevent dumping."

4 Conclusion

This cover search algorithm is applicable not only to examples but also to traditional rules if they are expressed in the form of translation patterns. Our position is that there is no essential difference between translation examples and translation rules, and that they can be handled in a uniform way; that is, a translation example is a special case of translation rules, whose node are lexical entries rather than categories. Moreover, one rule can have both lexical and categorical nodes. In line with this observation, we argue that there is a continues spectrum between translation examples and translation rules. Our cover search algorithm can efficiently handle both of them uniformly.

As mentioned in the introduction, three major technologies need to be developed in order to implement an EBT system: similarity calculation, example selection, and target construction. In this paper, we addressed the second one: an efficient method for selecting the most plausible set of translation examples for an input sentence. We have already developed a Japanese similarity calculator [7] for the similarity calculation and *rules combination transfer, or RCT* [6, 7] for the target structure construction. We are now developing the *Similarity-Driven Transfer System,* or *SimTran[7],* by combining the above three technologies.

References

[1] Nagao, M., "A Framework of a Mechanical Translation between Japanese and English by Analogy Principle," Elithorn, A. and Banerji, R. (eds.): *Artificial and Human Intelligence,* NATO 1984.

[2] Sadler, V., "Working with Analogical Semantics: Disambiguation Techniques in DLT," FORIS Publications, 1989.

[3] Sato, S., and Nagao, M., "Toward Memory-based Translation," Proc. of Coling '90.

[4] Sato, S., "Example-Based Translation Approach," Proc. of Int. Workshop on Fundamental Research for the Future Generation of NLP, July 23-24, 1991

[5] Sumita, E., Iida, H., and Kohyama, H., "Translating with Examples: A New Approach to Machine Translation," 3rd Int. Conf. on Theoretical and Methodological Issues in Machine Translation, 1990.

[6] Watanabe, H., "A Model of a Transfer Process Using Combinations of Translation Rules," Proc. of Pacific Rim Int. Conf. on AI '90.

[7] Watanabe, H., "A Similarity-Driven Transfer System," (will appear in) Proc. of Coling '92, 1992.