Optimizing Nearest Neighbor Retrieval by Similarity Template and Retrieval Query Generation

Takehito Utsuro
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama-cho, Ikoma-shi, Nara, 630-01, JAPAN
phone : +81-7437-2-5242 fax : +81-7437-2-5249
email: utsuro@is.aist-nara.ac.jp

Abstract

The nearest neighbor algorithm is the most basic class of techniques in the sub-fields of machine learning such as *case-based reasoning (CBR)*, *memory-based reasoning (MBR)*, and *instance-based learning (IBL)*. In the nearest neighbor algorithm, the computational cost of example retrieval is one of the most important issues. This paper proposes a novel technique for optimizing the nearest neighbor algorithm. Its basic idea is based on the use of *similarity template*, which is a data structure that enumerates all the possible patterns of calculating similarity between two examples. In the method, the nearest neighbor retrieval process is optimized by generating retrieval queries from an input and similarity templates in a certain order. Its major advantages are as follows: 1) unlike the hierarchical organization of memory, it is guaranteed that examples with the greatest similarity value according to the given similarity measure are retrieved, 2) it is easy to add new examples to the example database, 3) without any expensive hardware such as massively parallel computers, nearly constant time nearest neighbor retrieval can be achieved, independently of the number of examples in the database.

1 Introduction

In the field of machine learning (ML), most traditional learning methods form some abstraction from the experience and they store this structure in their memory. However, in recent years there has been growing interest in methods that store instances (examples or cases) in the memory and apply this specific knowledge directly to new situations. In the field of ML, *case-based reasoning (CBR)* [Kolodner, 1993], *memory-based reasoning (MBR)* [Stanfill and Waltz, 1986], and *instance-based learning (IBL)* [Aha *et al.*, 1991] are in this approach. In CBR, MBR, and IBL, the most basic class of techniques is called *the nearest neighbor algorithm*. It originated in the field of pattern recognition [Cover and Hart, 1967] and has been applied to many classification tasks. In the nearest neighbor algorithm, each training example is simply stored in the memory. Given a new test example, the algorithm finds the stored training example that is nearest according to some similarity measure, records the class of the retrieved example, and predicts that the new example will have the same class.

In the nearest neighbor algorithm, the computational cost of example retrieval is one of the most important issues, especially when the number of examples in the database becomes larger. The way of coping with this issue differs from one approach to another, and there exist at least two major approaches, namely, *hierarchical organization of memory* and *parallel search of flat memory*. Most practical CBR systems on serial computers (e.g., MEDIATOR [Kolodner and Simpson, 1989] and REMIND CBR Shell [Cognitive Systems, 1992]) adopted hierarchical organization of memory, while MBR [Stanfill and Waltz, 1986] and the massively parallel example-based natural language processing (NLP) approach (e.g., [Sumita *et al.*, 1993]) adopted parallel search of flat memory on massively parallel computers.

In approaches that adopted hierarchical organization of memory, such as a discrimination network and a decision tree, the case library is hierarchically partitioned using several features as indices, and according to the hierarchy, the search in the case library is guided and the search space is limited. When retrieving similar cases from the case library, cases satisfying the constraints of several indices are efficiently retrieved. Then, the nearest neighbor algorithm is used for ranking retrieved cases according to some similarity measure. Although this approach is adopted in many practical CBR systems on serial computers, it has several disadvantages. One of the most significant disadvantages is that the memory search and the ranking of cases are separate. Similar cases are searched within the hierarchically organized memory by checking several features as indices. During the search, however, the ranking of cases according to some similarity measure is not considered at all, but it is considered afterwards. Since the cases are not searched by checking all the features but only a few, there is no guarantee that a good case will not be missed. Another disadvantage is that the adding of cases is a complex operation and it is hard to keep the hierarchical memory optimal as cases are added.

On the other hand, in approaches that adopted parallel search of flat memory such as MBR and massively parallel example-based NLP, the similarity function is applied in parallel to all the examples in the example database, and all those examples are ranked according to the similarity values and the best matches are returned. In this approach, the whole library is searched and thus accuracy of retrieval is dependent only on the similarity measure. It is easy to add new examples to the database, since examples are stored in the database without any hierarchy. However, expensive hardware is needed and this is one of major disadvantages in this approach.

This paper proposes a novel technique for overcoming the problem of the computational cost of example retrieval in the nearest neighbor algorithm. The basic idea of the proposed method is based on the use of *similarity template*. A similarity template is a data structure which enumerates all the possible patterns of calculating similarity between two examples and it is independent of each example. The system contains a set of all the possible similarity templates and it generates a retrieval query from an input and a similarity template. Using the generated retrieval queries, the example space can be structured around the input as in Figure 1. In the structured example space, the nearest neighbor examples are located at the positions closest to the input. In this framework, the nearest neighbor retrieval process can be *optimized* by generating retrieval queries in a certain order. This optimization problem is a problem of finding a retrieval query which retrieves the nearest neighbor of the input as quickly as possible. In this paper, we describe two distinct strategies for optimizing the retrieval query generation process: *linear search along similarity value ordering* and *binary search along subsumption ordering of retrieval queries and similarity value ordering*. Compared with the hierarchical memory organization techniques, in which the memory search and the ranking of cases (or examples) are separate, the proposed method integrates these two in an optimal way, by directly searching the nearest neighbor(s) according to the structure of the example space.

The proposed method combines the advantages of both of the previous two approaches, i.e., hierarchical organization of memory and parallel search of flat memory. The major advantages are as follows: 1) it is guaranteed that examples with the greatest similarity value according to the given similarity measure are retrieved, 2) it is easy to add new examples to the example database, 3) without any expensive hardware such as massively parallel computers, nearly constant time nearest neighbor retrieval can be achieved, independently of the number of examples in the database.

In the previous paper [Utsuro *et al.*, 1994], we presented the method for optimizing the nearest neighbor retrieval only specifically in the case of retrieving Japanese surface case structures. In this paper, we provide more general motivations to our method by extending it in the context of ML, CBR, and MBR. The remainder of the paper is organized as follows.

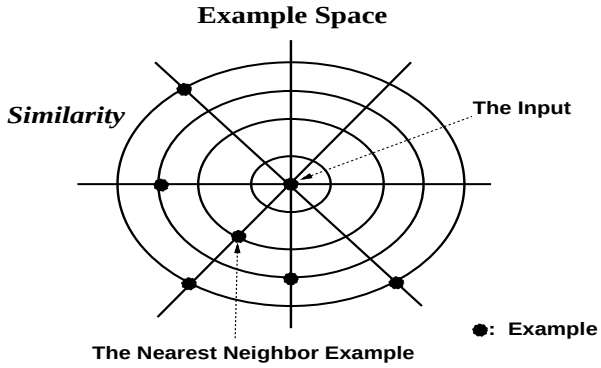

Figure 1: Structured Example Space

First, in section 2, we describe the basic idea of the proposed method in its most general form, without giving any specific definitions to the data structure of an example and to the similarity measure. Then, in section 3, we apply the optimization method to the standard data structure and the similarity measure which are most commonly used in the nearest neighbor retrieval in CBR and example-based NLP, namely, the set of feature-value pairs and the weighted sum of similarities for individual features. We describe a technique for applying our method to the general case of the data structure and the similarity measure, and then, as an example, we describe the retrieval of Japanese surface case structures with specific definitions of the data structure and the similarity measure. We also evaluate our method by showing that nearly constant time nearest neighbor retrieval is achieved, independent of the number of examples in the database.

2 The Basic Idea

This section presents the basic idea of optimizing nearest neighbor retrieval using similarity templates and retrieval query generation in its most general form, without giving any specific definitions to the data structure of an example and to the similarity measure.

2.1 Similarity Measure and Nearest Neighbor Retrieval

First, supposing that E be the set of all the examples, we introduce the similarity calculation function sim from $E \times E$ to R (the set of real numbers) which gives the similarity value between two examples:

$$sim: \ E \times E \mapsto R$$

Next, let Eg be the set of examples in the example database. Retrieval of the nearest neighbor(s) of the input e_{in} can then be regarded as the retrieval of the set $Eg_{max}(e_{in})$ of

examples which give the greatest similarity:

$$Eg_{max}(e_{in}) \quad = \quad \left\{ e \mid e \in Eg, sim(e, e_{in}) = \max_{e} sim(e, e_{in}) \right\}$$

2.2 Similarity Template and Similarity Calculation

The key idea of optimizing the nearest neighbor retrieval is the use of *similarity template*. First, we assume that a similarity template is definable only when the similarity measure satisfies the following two requirements:

Requirements on the Similarity Measure

1. Similarity can be regarded as a function of factors in the examples. Each example contains only a small number of factors.
2. Each factor varies over a few discrete values. If a factor varies over continuous values, these values can be transformed into a few discrete values.

A similarity template is now defined as a data structure which enumerates all the possible patterns of calculating similarity between two examples, and it is independent of each example. The left half of Figure 2 shows the idea of similarity calculation using similarity templates. Given any element e_1 from the example set 1 and any element e_2 from the example set 2, the result of their similarity calculation becomes the similarity template 1. Each similarity template has a similarity value which corresponds to the similarity between the two examples, and the similarity templates $1 \sim 3$ in the figure have the same similarity value.

More formally, supposing that T is the set of all similarity templates, the similarity calculation function sim can be defined as a composite function $sim_t \circ tmpl$, where $tmpl$ is a function from $E \times E$ to T which gives a similarity template of two examples, and sim_t is a function from T to R which gives the similarity value of a similarity template:

$$tmpl : \ E \times E \mapsto T$$

$$sim_t : \ T \mapsto R$$

$$sim = sim_t \circ tmpl : \ E \times E \mapsto R$$

2.3 Retrieval Query Generation

A retrieval query can be considered as a set of constraints on the examples to be retrieved by the query. A retrieval query is generated from an input and a similarity template by the

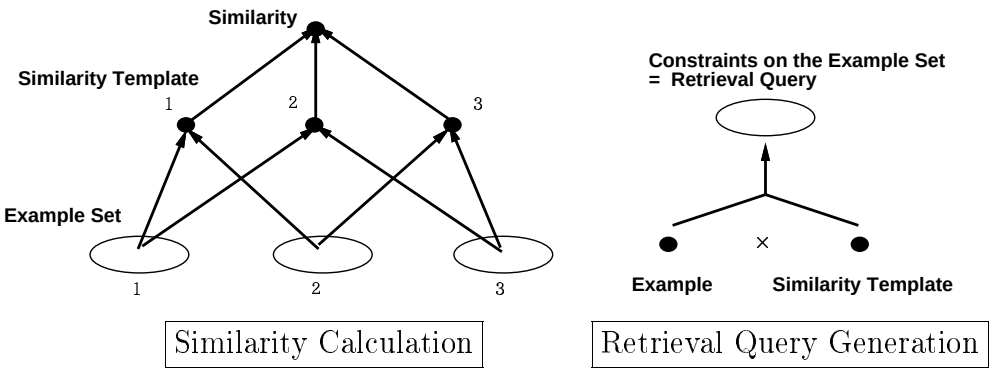

Figure 2: The Basic Idea of Similarity Template

retrieval query generation function *qgen*. *qgen* is a function from $E \times T$ to Q, where Q is the set of all retrieval queries. The right half of Figure 2 shows the idea of retrieval query generation from an input and a similarity template. Given an input e_{in} and a similarity template t, *qgen* returns a retrieval query q:[1]

$$qgen: \ E \times T \mapsto Q$$

$$q \ = \ qgen(e_{in}, t)$$

2.4 Overall Framework

Figure 3 shows the overall framework of the optimized nearest neighbor retrieval using similarity templates and retrieval queries. The system contains only the set of similarity templates, the indexed retrieval module, and the example database. It does not contain any retrieval queries in advance. A similarity template is a data structure which is independent of each example, while a retrieval query is generated from the input and a similarity template, and it is thus dependent on the input. For a given input, retrieval queries are generated in a certain order using similarity templates. Then, examples which satisfy a retrieval query are quickly retrieved from the example database using the indexed retrieval module. The optimization problem of the nearest neighbor retrieval is considered as a problem of optimizing the retrieval query generation process. This is a problem of finding a retrieval query which retrieves the nearest neighbor of the input as quickly as possible, and to do this, it must be guaranteed that the example database does not contain any examples which have a greater similarity value.

[1] If $Eg(q)$ is the set of examples which satisfy the constraints of the retrieval query q, it is not necessary the case that the similarity template of e and e_{in}, i.e. $tmpl(e_{in}, e)$, becomes t for all the examples e in $Eg(q)$. This is so because a retrieval query q represents constraints on the examples and some example e which satisfies q could also satisfy much stronger constraints. In those cases, the similarity template of e and e_{in} would become much *stronger* than the similarity template t. In fact, this is true for the definition of the retrieval query in section 3.1.

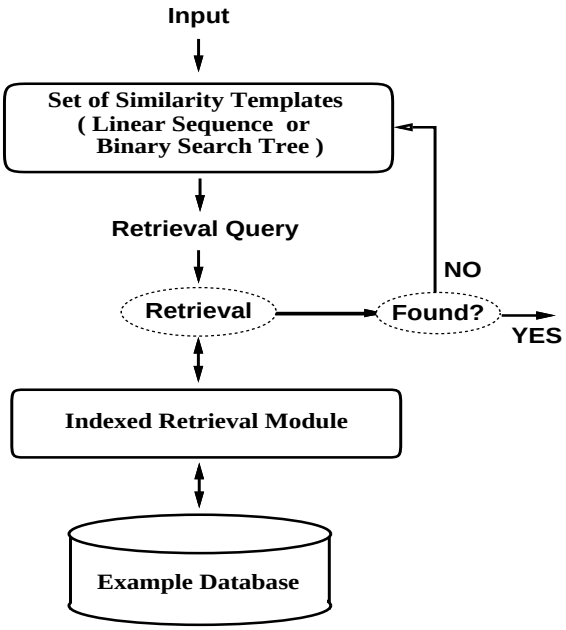

Figure 3: The Framework of the Optimized Nearest Neighbor Retrieval

2.5 Optimizing the Nearest Neighbor Retrieval

This section describes two distinct strategies for optimizing the retrieval query generation process: *linear search along similarity value ordering* and *binary search along subsumption ordering of retrieval queries and similarity value ordering.*

2.5.1 Linear Search along Similarity Value Ordering

This is a simple linear search strategy which starts from a retrieval query for the greatest similarity value and continues to those for lower similarity value, gradually decreasing the similarity value, until it finds at least one example. In Figure 1, linear search corresponds to the search from the input outward until at least one nearest neighbor is found.

For the linear search, the set of all similarity templates is first divided into a sequence $T_1, \ldots, T_n$ which is totally ordered by the similarity value:

$$\begin{aligned} \forall t \in T_i, \forall t' \in T_j, \qquad & i < j \;\Rightarrow\; sim_t(t) > sim_t(t') \\ & i = j \;\Rightarrow\; sim_t(t) = sim_t(t') \end{aligned}$$

From the sequence $T_1, \ldots, T_n$ of the sets of similarity templates, a sequence $Q_1, \ldots, Q_n$ of the sets of retrieval queries is constructed by collecting retrieval queries generated from each similarity template as below:

$$Q_i \;=\; \left\{ q \mid \exists t \in T_i, q = qgen(e_{in}, t) \right\} \; (i = 1, \ldots, n) \qquad (1)$$

Then, starting from the first set Q_1 of the sequence $Q_1, \ldots, Q_n$, the linear search process retrieves examples using retrieval queries in each set until at least one example is retrieved. If at least one example is retrieved using one of retrieval queries in the set Q_i, then the set $Eg(Q_i)$ of examples retrieved by all the retrieval queries in Q_i is the set of nearest neighbor examples.[2]

2.5.2 Binary Search along Subsumption Ordering of Retrieval Queries and Similarity Value Ordering

It can happen that the nearest neighbor of an input has a small similarity value and it is inefficient to search the nearest neighbor linearly along similarity value ordering. Compared with the linear search strategy, the binary search strategy makes efficient retrieval possible independently of whether the nearest neighbor has a high similarity value or not.

The basic idea of the binary search strategy is to partition the example space using retrieval queries according to the subsumption relation of retrieval queries and the similarity value. Subsumption relation of retrieval queries corresponds to the subsumption relation of the example sets retrieved by the queries. This means that if constraints in a retrieval query q_1 subsume those in another retrieval query q_2, then the set $Eg(q_1)$ of the examples retrieved by q_1 subsumes the set $Eg(q_2)$ of the examples retrieved by q_2. The example space is partitioned and structured using subsumption relation of the example sets, and also using similarity value ordering. With the structured example space, it becomes possible to binary-search the nearest neighbor(s) of the input.

Totally Ordered Example Sets

In the same way as for the linear search, also for the binary search, the set of all similarity templates is first divided into a sequence $T_1, \ldots, T_n$, and from this sequence, a sequence $Q_1, \ldots, Q_n$ of the sets of retrieval queries is constructed by Formula 1 in section 2.5.1. However, now the sequence $Eg(Q_1), \ldots, Eg(Q_n)$ of the example sets retrieved by the retrieval queries in each set Q_i $(i = 1, \ldots, n)$ has to satisfy the following two requirements of total ordering:

1. **Subsumption Relation of Example Sets**

   The sequence $Eg(Q_1), \ldots, Eg(Q_n)$ is totally ordered by the subsumption relation as in Figure 4:

   $$Eg(Q_1) \subseteq \cdots \subseteq Eg(Q_n)$$

[2]It is not necessary to construct all the sets $Q_1, \ldots, Q_n$ in advance before starting example retrieval using retrieval queries in each set. It is sufficient to construct a set Q_i of retrieval queries when no examples are retrieved using the preceding sets $Q_1, \ldots, Q_{i-1}$. This also holds for the binary search in the next section.

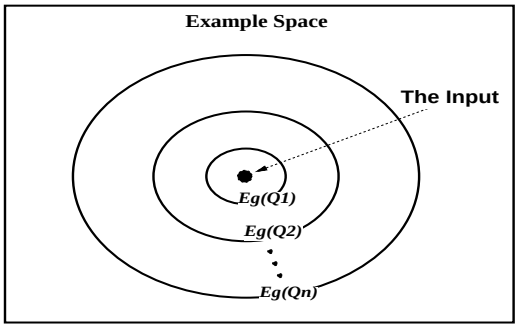

Figure 4: Subsumption Relation of Example Sets

2. **Similarity Value of Each Example**

   The examples in the inner sets have higher similarity values than the examples in the outer sets:

$$Eg(Q_i) \subseteq Eg(Q_j),\ \forall e_i \in Eg(Q_i), \forall e_j \in Eg(Q_j),$$
$$(\ e_j \notin Eg(Q_i) \quad \Rightarrow \quad sim(e_{in}, e_i) > sim(e_{in}, e_j)\ )$$

If it is possible to divide the set of all similarity templates into a sequence $T_1, \ldots, T_n$ which satisfies the two requirements above, then the following binary search strategy for the nearest neighbor retrieval becomes applicable.

Binary Search of the Nearest Neighbor

With the totally ordered sequence of example sets, the nearest neighbor(s) with the greatest similarity value exist in the innermost non-empty set of examples. The innermost non-empty set can be efficiently found by the binary search procedure using the binary search tree in Figure 5. The binary search tree in Figure 5 is constructed from the totally ordered sequence $Eg(Q_1), \ldots, Eg(Q_n)$ of example sets, in which each node is a set of examples $Eg(Q_i)$. The following is the binary search strategy for finding the nearest neighbor(s) of the input.[3]

1. At any non-leaf node $Eg(Q_i)$, if $Eg(Q_i)$ contains at least one example then the search process goes down to the left child $Eg(Q_j)$. Otherwise, the search process goes down to the right child $Eg(Q_k)$.

[3]In this section, we describe the binary search strategy using the binary search tree of *the example sets* just for explanation. In the framework described in section 2.4, the system contains only the binary search tree of *the sets of similarity templates* but not the binary search tree of *the example sets* like the one in Figure 5. As we mentioned in the previous section, it is sufficient to construct the set Q_i of retrieval queries only when the search process has proceeded to T_i.

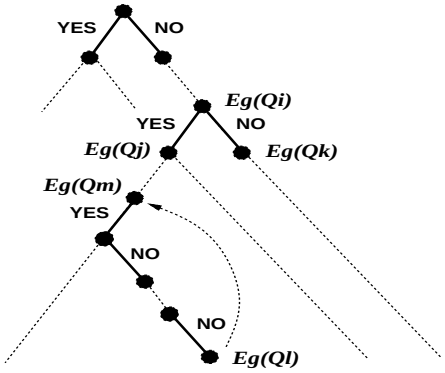

Figure 5: Binary Search Tree of the Nearest Neighbor Retrieval

2. At any leaf node $Eg(Q_l)$, if $Eg(Q_l)$ contains at least one example then $Eg(Q_l)$ is the innermost non-empty set. Otherwise, the innermost non-empty set is the lowest non-empty ancestor $Eg(Q_m)$ of $Eg(Q_l)$.

3. Construct the set $Eg_{max}(e_{in})$ of the nearest neighbor(s) from the innermost non-empty set.

3 An Example

This section applys the optimization method to the standard data structure and the similarity measure which are most commonly used in the nearest neighbor retrieval in CBR and example-based NLP, namely, the set of feature-value pairs and the weighted sum of similarities for individual features. We then define similarity template for the definition of the similarity measure, and also show how to optimize the nearest neighbor retrieval based on the binary search strategy given in section 2.5.2. After that, we describe the retrieval of Japanese surface case structures as an example. Finally, we evaluate the system size and the computational cost of the overall framework.

3.1 Definitions

Data Structure of Example

We represent an example e as an n-tuple of feature-value pairs:

$$e \;=\; \left\langle\, \langle f_1, v_1\rangle, \ldots, \langle f_n, v_n\rangle \,\right\rangle$$

Similarity Measure

Let w_i be the weight for the i-th feature and $s(v_{1i}, v_{2i})$ be the similarity of the i-th values of the examples e_1 and e_2. Then, the similarity $sim(e_1, e_2)$ of e_1 and e_2 is defined as the weighted sum of similarities for individual features:

$$sim(e_1, e_2) \quad = \quad \Big(\sum_{i=1}^{n} w_i \times s(v_{1i}, v_{2i}) \Big) \Big/ \sum_{i=1}^{n} w_i \tag{2}$$

We assume that the weights w_i $(i = 1, \ldots, n)$ and the similarity function $s(v_{1i}, v_{2i})$ for individual features satisfy the two requirements on the similarity measure stated in section 2.2.

Similarity Template

For the similarity measure defined in Formula 2, we define a similarity template as an n-tuple of the similarities for individual features. The similarity function sim_t for similarity template is also defined from the similarity function sim of examples in Formula 2:

$$\begin{aligned} t \quad &= \quad \langle s_1, \ldots, s_n \rangle \\ sim_t(t) \quad &= \quad \Big(\sum_{i=1}^{n} w_i \times s_i \Big) \Big/ \sum_{i=1}^{n} w_i \end{aligned} \tag{3}$$

Retrieval Query

A retrieval query q is defined as an n-tuple of the constraints on individual feature values:

$$q \quad = \quad \Big\langle \, \langle f_1, V(v_1, s_1) \rangle, \ldots, \langle f_n, V(v_n, s_n) \rangle \, \Big\rangle \tag{4}$$

In this definition, each constraint is denoted as $V(v_i, s_i)$, which means that the i-th value v_{ri} of the example e_r to be retrieved has to satisfy the similarity constraint that v_{ri} and v_i have the similarity value greater than or equal to s_i:

$$s(v_{ri}, v_i) \quad \geq \quad s_i$$

As described in section 2.3, a retrieval query is generated from the input e_{in} and the similarity template t by the function $qgen$. If e_{in} is represented as $\langle \langle f_1, v_1 \rangle, \ldots, \langle f_n, v_n \rangle \rangle$ and t is given as Formula 3, the retrieval query q generated by $qgen(e_{in}, t)$ is given as Formula 4.

3.2 Total Ordering of the Sets of Similarity Templates

For the binary search strategy given in section 2.5.2, we first have to divide the set of all similarity templates into a sequence $T_1, \ldots, T_n$ which satisfies the requirements of *subsumption relation of example sets* and *similarity value of each example*. In the case of the similarity measure and the similarity template in this section, these two requirements can be restated using the subsumption relation and the similarity value of similarity templates.

Subsumption Relation of Similarity Templates and Example Sets

First, we define the subsumption relation of similarity templates and derive the subsumption relation of example sets from that. The subsumption relation of similarity templates can be regarded as the subsumption relation of the similarity constraints on individual feature values. Let t and t' be similarity templates $\langle s_1, \ldots, s_n \rangle$ and $\langle s'_1, \ldots, s'_n \rangle$ respectively. Then the subsumption relation $\prec_t$ of similarity templates is defined as follows: $t \prec_t t'$ (t subsumes t') is true if and only if each similarity s_i is smaller than or equal to s'_i for all the features and t is not identical to t', i.e.

$$t \prec_t t' \stackrel{\text{def}}{\iff} s_i \leq s'_i \ \ (i = 1, \ldots, n) \ \text{ and } t \text{ is not identical to } t'$$

Then, assume that t subsumes t' and let e_{in} be an input and $q = qgen(e_{in}, t)$ $(= \langle \langle f_1, V(v_1, s_1) \rangle, \ldots, \langle f_n, V(v_n, s_n) \rangle \rangle)$ and $q' = qgen(e_{in}, t')$ $(= \langle \langle f_1, V(v_1, s'_1) \rangle, \ \ldots, \ \langle f_n, V(v_n, s'_n) \rangle \rangle)$ be the retrieval queries generated by $qgen$. Then, with the definition above, it is obvious that q represents a weaker constraint on the examples to be retrieved than q', since each similarity constraint on its feature values is weaker than or equal to the similarity constraint on q''s feature values. Thus, letting $Eg(q)$ and $Eg(q')$ be the example sets retrieved by q and q' respectively, we can conclude that $Eg(q)$ subsumes $Eg(q')$:

$$t \prec_t t' \ \Rightarrow \ Eg(q) \supseteq Eg(q') \tag{5}$$

Total Ordering of the Sets of Similarity Templates

Next, we show how to divide the set of all similarity templates into a sequence $T_1, \ldots, T_n$ which satisfies the requirements of *subsumption relation of example sets* and *similarity value of each example* in section 2.5.2. We give the division into a sequence $T_1, \ldots, T_n$ using *the subsumption relation of similarity templates* and *the similarity value of similarity template*. Then, we show this division satisfies the two requirements in section 2.5.2.

1. **Subsumption Relation of Similarity Templates**

   We assume that the sequence $T_1, \ldots, T_n$ satisfies the condition that for each t_i in T_i, there exists a t_j in T_j $(i < j)$ such that t_j subsumes $(\prec_t)$ t_i:

   $$i < j \ \Rightarrow \ \forall t_i \in T_i, \ \exists t_j \in T_j, \ t_i \succ_t t_j \tag{6}$$

   As described in section 2.5.2, a sequence $Q_1, \ldots, Q_n$ of the sets of retrieval queries is constructed from the sequence $T_1, \ldots, T_n$ using Formula 1. From Formulas 5 and 6 above, it can be concluded that the sequence $Q_1, \ldots, Q_n$ satisfies the following:

   $$i < j \ \Rightarrow \ \forall q_i \in Q_i, \ \exists q_j \in Q_j, \ Eg(q_i) \subseteq Eg(q_j)$$

Thus, $Eg(Q_j)$ subsumes $Eg(Q_i)$ $(i < j)$. It is obvious from this that the sequence $Q_1, \ldots, Q_n$ satisfies the requirement of the subsumption relation of examples sets in section 2.5.2:

$$Eg(Q_1) \subseteq \cdots \subseteq Eg(Q_n)$$

2. **Similarity Value of Similarity Template**

We assume that the sequence $T_1, \ldots, T_n$ satisfies the condition that for each t_i in T_i and t_j in T_j $(i < j)$, $sim_t(t_i)$ is greater than $sim_t(t_j)$:

$$i < j \;\Rightarrow\; \forall t_i \in T_i,\ \forall t_j \in T_j,\ sim_t(t_i) > sim_t(t_j)$$

In order to prove that the sequence $T_1, \ldots, T_n$ satisfies the requirement of the similarity value of each example in section 2.5.2, it is sufficient to prove the requirement for the case of $j = i + 1$. For each e_i in $Eg(Q_i)$ and for each e_{i+1} in $Eg(Q_{i+1})$, if e_{i+1} is not in $Eg(Q_i)$, then there exist a t_i in T_i and a t_{i+1} in T_{i+1} such that $e_i \in Eg(qgen(e_{in}, t_i))$ and $e_{i+1} \in Eg(qgen(e_{in}, t_{i+1}))$, and for each t_i in T_i, $e_{i+1} \notin Eg(qgen(e_{in}, t_i))$. From this, the following relation of the similarity values of examples and the similarity values of similarity templates holds[4]:

$$sim(e_{in}, e_i) \;\geq\; \min_{t_i \in T_i} sim_t(t_i) \;>\; sim(e_{in}, e_{i+1}) \;\geq\; \min_{t_{i+1} \in T_{i+1}} sim_t(t_{i+1})$$

Thus $sim(e_{in}, e_i)$ is greater than $sim(e_{in}, e_{i+1})$.

Then, as we presented in section 2.5.2, it becomes possible to binary-search the nearest neighbor(s) of the input using the sequence $T_1, \ldots, T_n$ of sets of similarity templates.

3.3 Retrieval of Japanese Surface Case Structures

As an example of optimizing the nearest neighbor retrieval, this section describes retrieval of Japanese surface case structures.[5] We represent an example of a Japanese surface case structure as a 3-tuple of feature-value pairs[6]:

$$e \;=\; \left\langle\ \langle verb, Sem_v\rangle, \langle subj, Sem_s\rangle, \langle obj, Sem_o\rangle\ \right\rangle$$

[4]Note that if $e \in Eg(qgen(e_{in}, t))$ is true, then $sim(e_{in}, e) \geq sim_t(t)$ is true. And also note that if $e_{i+1} \notin Eg(qgen(e_{in}, t_i))$ is true for all the t_i in T_i, then $\min_{t_i \in T_i} sim_t(t_i) > sim(e_{in}, e_{i+1})$ is true.

[5]In example-based NLP, retrieval of similar surface case structures is a basic technique in the tasks such as example-based target word selection in machine translation [Sato and Nagao, 1990] and example-based case structure analysis [Kurohashi and Nagao, 1993].

[6]In [Utsuro *et al.*, 1994], we adopted much more practical data structure of examples and similarity measure for retrieval of Japanese surface case structures, and also discussed how to optimize the nearest neighbor retrieval.

verb, *subj*, and *obj* are features for the verb, subject (nominative case or *ga* case), and object (accusative case or *wo* case) of the surface case structure, respectively. The values Sem_v, Sem_s, and Sem_o represent the semantic categories of the verb, subject noun, and object noun, respectively. For representing the semantic categories, we use an on-line Japanese thesaurus called *Bunrui Goi Hyou* (BGH) [NLRI, 1964]. BGH has a six-layered abstraction hierarchy and more than 60,000 Japanese words are assigned to the leaves.

The similarity measure *sim* for this data structure of Japanese surface case structures is defined on the basis of the similarity measure in section 3.1. The weights w_1, w_2, and w_3 are set as 1 for simplicity. In the following definition, $s(Sem_{1v}, Sem_{2v})$, $s(Sem_{1s}, Sem_{2s})$, and $s(Sem_{1o}, Sem_{2o})$ represent the similarities of corresponding semantic categories:

$$sim(e_1, e_2) \;=\; \Big(\; s(Sem_{1v}, Sem_{2v}) + s(Sem_{1s}, Sem_{2s}) + s(Sem_{1o}, Sem_{2o}) \; \Big) \;\Big/\; 3$$

We define the similarity $s(Sem_1, Sem_2)$ of two semantic categories Sem_1 and Sem_2 as a monotonically increasing function of the most specific common layer $mscl(Sem_1, Sem_2)$ of Sem_1 and Sem_2 in the thesaurus as below:

$mscl(Sem_1, Sem_2)$	1	2	3	4	5	6	exact match
$s(Sem_1, Sem_2)$	0	5	7	8	9	10	11

For example, the similarity of the surface case structures e_1 and e_2 of Examples 1 and 2 is calculated as follows. First, the surface case structures e_1 and e_2 are given as below:

Example 1

kare - ga	hon - wo	kau
he - NOM	*book - ACC*	*buy*

(He buys a book.)

Example 2

kanojo - ga	nooto - wo	kau
she - NOM	*notebook - ACC*	*buy*

(She buys a notebook.)

$$\begin{aligned} e_1 &= \Big\langle \; \langle verb, Sem_{buy} \rangle, \langle subj, Sem_{he} \rangle, \langle obj, Sem_{book} \rangle \; \Big\rangle \\ e_2 &= \Big\langle \; \langle verb, Sem_{buy} \rangle, \langle subj, Sem_{she} \rangle, \langle obj, Sem_{nb} \rangle \; \Big\rangle \quad (nb{=}notebook) \end{aligned}$$

The similarity values for the semantic categories in the thesaurus are as below:

$$s(Sem_{buy}, Sem_{buy}) = 11, \;\; s(Sem_{he}, Sem_{she}) = 10, \;\; s(Sem_{book}, Sem_{nb}) = 9$$

Finally, the similarity of e_1 and e_2 is calculated as below:

$$sim(e_1, e_2) \;=\; (11 + 10 + 9) \;/\; 3 \;=\; 10$$

For the similarity measure defined above, we define a similarity template as a 3-tuple of the similarity for individual features:

$$t \;=\; \langle s_v, s_s, s_o \rangle \quad \Big(\; s_v, s_s, s_o \in \{0, 5, 7, 8, 9, 10, 11\} \; \Big)$$

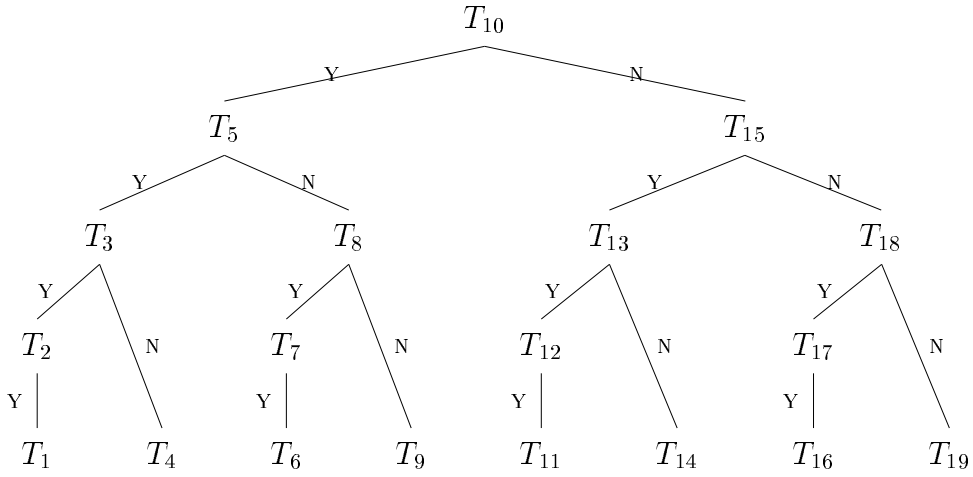

Figure 6: An Example of Binary Search Tree of the Sets of Similarity Templates

The similarity function sim_t for the similarity template and the retrieval query are defined on the basis of the defintions in section 3.1. In the similarity calculation of e_1 and e_2 above, the similarity template is $\langle 11, 10, 9 \rangle$.

Next, we divide the set of all similarity templates into a sequence $T_1, \ldots, T_n$ which satisfies the requirements of subsumption relation of similarity templates and similarity value of similarity template in section 3.2. The number of possible similarity templates is 343 ($=7^3$), and the set of all similarity templates is divided into a sequence of 19 subsets $T_1, \ldots, T_{19}$.[7] Figure 6 shows the binary search tree constructed from the sequence $T_1, \ldots, T_{19}$. As described in section 2.5.2, the arc labels "Y" and "N" correspond to the two possibilities whether or not there exists a similarity template in the parent node such that a retrieval query generated from this retrieves at least one example.

Suppose that the input is the surface case structure e_1 of Example 1 and the nearest neighbor in the example database is e_2 of Example 2. The process of binary search in the binary search tree of Figure 6 is then as follows. The similarity template of e_1 and e_2 is $t_{12} = \langle 11, 10, 9 \rangle$ and this is contained in T_4. The binary search process starts from the root node T_{10} and proceeds in the order of $T_{10} \rightarrow T_5 \rightarrow T_3 \rightarrow T_4$. Finally, a retrieval query q_{12} is generated from e_1 and t_{12} as below, and e_2 is retrieved by q_{12}:

$$q_{12} \;=\; \Big\langle\; \langle verb, V(Sem_{buy}, 11)\rangle, \langle subj, V(Sem_{he}, 10)\rangle, \langle obj, V(Sem_{book}, 9)\rangle \;\Big\rangle$$

In the process of the binary search, subsumption relation of similarity templates can be used for eliminating similarity templates which are useless for retrieving examples from the

[7] The length of the sequence of the subsets of similarity templates differs depending on the definition of the similarity measure: the longer the sequence, the more efficient the binary search is for the same number of similarity templates. Sometimes it can happen that there is no unique division of the set of all similarity templates.

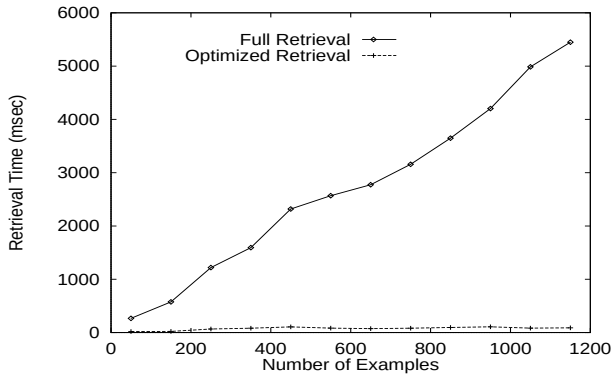

Figure 7: Retrieval Time per Number of Examples

example database.[8]

3.4 Evaluation

This section evaluates the system size and the computational cost of the overall framework of the nearest neighbor retrieval proposed in section 2.4, in the case of the retrieval of Japanese surface case structures presented in section 3.3. As an indexed retrieval module of section 2.4, we use a tree-structured index module constructed from the on-line Japanese thesaurus, which we call *sub-thesaurus* [Utsuro *et al.*, 1994]. For each feature, examples which satisfy the constraint $V(Sem, s)$ are collected for all the leaf semantic categories Sem in the thesaurus and all the similarity values s. Then, a sub-thesaurus for that feature is constructed as a sub-structure of the whole thesaurus in which each node corresponds to a set of examples that satisfy the constraint of the semantic category of the node. Using those sub-thesauri as the index module, examples which satisfy a retrieval query are obtained as examples which satisfy constraints on all the features. This retrieval can be done quickly in constant time independetly of the number of examples in the example database.

Next, let N be the size of the example database. The number of all similarity templates is independent of N and that of the indexed retrieval module, i.e., the set of sub-thesauri, is $O(N)$. Thus, the total order of the system size is $O(N)$. Finally, in order to evaluate the computational cost, we plot the computation time (in CPU time), increasing the number of examples N, and compare the result with *a full retrieval* program. The example database contains example Japanese surface case structures and both programs retrieve the most similar examples from the example database, given an input surface case structure. The full

[8]Formula 5 in section 3.2 represents the relation between the similarity template subsumption $\prec_t$ and the example set subsumption $\supseteq$. From this, we can conclude that if a retrieval query generated from a similarity template t retrieves no example, then any similarity template t' subsumed by t generates no retrieval query that would retrieve at least one example. Thus, we can eliminate all the similarity templates that are subsumed by t.

retrieval program calculates the similarity between the input and the example for all the examples in the example database, and retrieves the examples with the greatest similarity. Both programs are implemented in SICStus Prolog 2.1 on a SPARC station-10. Figure 7 illustrates the results. The computation time of the full retrieval program is proportional to N, while that of *the optimized retrieval* program is nearly constant. Thus, our optimized retrieval program achieved drastic improvement in decreasing computational cost compared with the full retrieval program.

4 Concluding Remarks

This paper proposed a novel technique for optimizing the nearest neighbor algorithm, based on the use of similarity template, which is a data structure to enumerate all the possible patterns of calculating similarity between two examples. The nearest neighbor retrieval process is optimized by generating retrieval queries from an input and similarity templates in a certain order.

Among previous works on CBR, [Shimazu *et al.*, 1993] took an approach relatively similar to ours. In [Shimazu *et al.*, 1993], given an input, retrieval queries are generated using Standard Query Language (SQL) and then nearest neighbors are retrieved from a commercially available relational data-base system (RDBMS). Their search strategy corresponds to our linear search in section 2.5.1. Compared with the formalization of this paper, they provided only a specific technique of generating SQL queries from an input and thus lacked any formal accounts of optimizing the nearest neighbor algorithm.

The set of feature-value pairs and the weighted sum of similarities for individual features, to which we applied our method in section 3, are most commonly used as the standard data structure and the similarity measure in CBR. Sometimes, however, it is not sufficient to assign only one set of weights, especially when the case library is used for several different purposes. If it is necessary to assign several different sets of weights dependening on the purpose of the reasoning, then we have to prepare several different sequences $T_1, \ldots, T_n$ of the sets of similarity templates according to the sets of weights.

References

[Aha *et al.*, 1991] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[Cognitive Systems, 1992] Cognitive Systems. *ReMind Developer's Reference Manual.* 1992.

[Cover and Hart, 1967] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[Kolodner and Simpson, 1989] J. Kolodner and R. Simpson. The MEDIATOR: Analysis of an early case-based problem solver. *Cognitive Science*, 13(4):507–549, 1989.

[Kolodner, 1993] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[Kurohashi and Nagao, 1993] S. Kurohashi and M. Nagao. Structural disambiguation in Japanese by evaluating case structures based on examples in case frame dictionary. In *Proceedings of the 3rd IWPT*, pages 111–122, 1993.

[NLRI, 1964] NLRI, (National Language Research Institute). *Word List by Semantic Principles*. Syuei Syuppan, 1964. (in Japanese).

[Sato and Nagao, 1990] S. Sato and M. Nagao. Toward memory-based translation. In *Proceedings of the 13th COLING*, volume 3, pages 247–252, 1990.

[Shimazu *et al.*, 1993] H. Shimazu, H. Kitano, and A. Shibata. Retrieving cases from relational data-bases: Another stride towards corporate-wide case-base systems. In *Proceedings of the 13th IJCAI*, pages 909–914, 1993.

[Stanfill and Waltz, 1986] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[Sumita *et al.*, 1993] E. Sumita, K. Oi, O. Furuse, H. Iida, T. Higuchi, N. Takahashi, and H. Kitano. Example-based machine translation on massively parallel processors. In *Proceedings of the 13th IJCAI*, pages 1283–1288, 1993.

[Utsuro *et al.*, 1994] T. Utsuro, K. Uchimoto, M. Matsumoto, and M. Nagao. Thesaurus-based efficient example retrieval by generating retrieval queries from similarities. In *Proceedings of the 15th COLING*, pages 1044–1048, August 1994.