JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1633-1653 (2018)
DOI: 10.6688/JISE.201811_34(6).0015

# An Efficient Mechanism for Compensating Vague Pattern Identification in Support of a Multi-Criteria Recommendation System

YI-CHUNG CHEN[1] AND CHIANG LEE[2]
[1]*Department of Industrial Engineering and Management*
*National Yunlin University of Science and Technology*
*Yunlin, 640 Taiwan*
[2]*Department of Computer Science and Information Engineering*
*National Cheng Kung University*
*Tainan, 701 Taiwan*
*E-mail: chenyich@yuntech.edu.tw; leec@mail.ncku.edu.tw*

A vague pattern usually makes the result of pattern identification specious. Most existing identification algorithms try to upgrade their identification accuracies by improving the clearness of the vague pattern. However, this improvement can be limited due to the poor quality of the pattern itself. Hence, the identification result can still be untrustworthy and thus a user needs to repeat the algorithm to find another possible answer, which can be quite time-consuming. In this paper, we propose a novel pattern recommendation mechanism which is able to obtain multiple highly possible answers from a large datapool based on a given vague pattern. By using the identification algorithm only one time, a user can select a correct identification answer from these candidates given by the recommendation system. Three strategies are proposed in this paper. Experiments are performed to demonstrate the effectiveness and efficiency of the proposed mechanism.

***Keywords*:** pattern identification, vague pattern, recommendation mechanism, multi-criteria search, R-tree

## 1. INTRODUCTION

The pattern identification problem has attracted considerable attention in recent years [7, 20]. Most existing identification algorithms first retrieve the representative *d* features from a pattern datapool, such as in PCA [5, 13], LDA [2], ICA [2, 5], NN [4, 15], then find from the datapool a pattern similar to the query pattern. For example, Table 1 is a face feature datapool, which records the distance between two eyes and the width of the mouth of each face. Fig. 1 is the result of mapping these patterns into a coordinates system. Each point in this figure represents a face, with its coordinates representing the distance between two eyes and the width of mouth. Assume that a police has a face picture Q of a thief, which is acquired from a surveillance system, and is trying to find a matching face from a datapool by using an identification algorithm. The algorithm first retrieves the distance between two eyes and the width of the mouth of Q (say, (3.95, 4.95)). Then, the algorithm finds that C(3.9, 4.9) is the closest pattern to Q, as shown in Fig. 1. Hence, C has the highest possibility to be the thief.

However, if Q is a vague picture, the identification result can be incorrect. The am-

Received December 4, 2015; revised October 5, 2016; accepted June 27, 2018.
Communicated by Vincent S. Tseng.

**Table 1. The face datapool.**

| Face | Distance between eyes (cm) | Width of mouth (cm) |
|---|---|---|
| A | 3.95 | 5.4 |
| B | 4.3 | 5 |
| C | 3.9 | 4.9 |
| D | 4.2 | 5.3 |
| E | 4.8 | 5.5 |
| F | 4.5 | 4.4 |
| G | 5 | 4.8 |
| H | 4.1 | 4.1 |

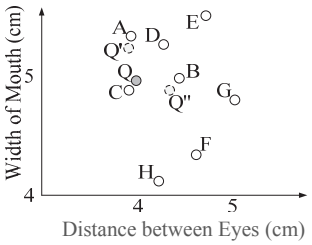


Fig. 1. The face datapool.

biguity may be caused by low resolution of the surveillance system [7, 10, 11], a long distance between Q and the monitor [3, 21], or the motion blur on the image [1, 11, 14], *etc.* The extent of uncertainty in different features can be quite different. For example, the image of eyes of Q may be clear, but the mouth is blurred. If the correct value in this case is (3.7, 5.4) (*i.e.*, Q′ in Fig. 1), then the answer should be A instead of C. On the other hand, if the image of eyes of Q is seriously affected by the motion blur but the part of mouth is not so that the correct value is (4.6, 4.8) (*i.e.*, Q″ in Fig. 1), then the answer should be B rather than A. These two cases indicate two problems of the existing identification algorithms; (1) When ambiguity exists, the result returned by an identification algorithm can be incorrect. Hence, the user needs to operate the identification algorithm again to find another possible answer, which is quite awkward and time consuming; (2) Different degree of ambiguity in different features may cause the search result to be completely different, which is not dealt with in the existing identification algorithms.

The design philosophy of the most existing identification algorithms in dealing with an ambiguous pattern is to improve the quality of the query pattern and to lower the errors of the captured features. However, we must understand that no matter what quality improvement method is used, the resulting improvement may be limited due to the poor quality of the target pattern, which then results in numerous errors in the features captured by the feature extraction algorithm and the results of the feature identification algorithm. Thus, the conventional approach of increasing recognition rates by improving quality has some room for improvement. For instance, if the query pattern to be dealt with is a signal, such as in the speech identification [19] and the activity identification problem [9, 20], a filter is usually used before the algorithm to sift out the noises. Nevertheless, we know that each filter has its own capability limits, so signal quality cannot be effectively increased when the signal contains a lot of noise. If the query pattern is an image and its resolution is low, Gunturk *et al*. [7] and Hennings-Yeomans *et al*. [10] propose to construct a super-resolution image from multiple low resolution images. Still, we must understand that in conditions with lower image resolution, many objects were blurry to begin within their original images and thus unrecognizable no matter how they are processed. In view of this, this study proposed a concept unlike those of conventional methods to effectively increase the recognition rate of target patterns.

In this paper, we propose another philosophy in resolving this problem, which is to retrieve multiple highly possible answers (*i.e.*, patterns) and return to the user. The user will decide from these answers which one(s) will be the correct answer. The user may also issue a threshold $k$ before the search process to limit the number of possible answers. The mechanism devised in this paper is called a multi-criteria based $k$-pattern recommendation scheme. This mechanism is used "after" an existing identification algorithm.

Assume that the query pattern Q is the input to the identification algorithm, which uses $d$ numerical features in its process. The result pattern found by the identification algorithm is $r$. The proposed mechanism then utilizes $r$ and the $d$ numerical features to evaluate the score of each pattern in the datapool. The patterns with a score smaller than $k$ are returned to the user. In our design, the smaller the score of a pattern, the more likely that this pattern is the answer of the query. As far as we know, this paper presents a first attempt to resolve the identification problem by using a recommendation scheme.

The multi-criteria based $k$-pattern recommendation scheme has a few notable features. First, by operating the identification algorithm only one time, the proposed mechanism can return all possible answers of a query pattern. Hence, the cost incurred by repeatedly operating the identification algorithm as in the traditional manner can be avoided. Second, the mechanism is able to find suitable answers for a query pattern that has different degree of ambiguity in different features. That is, even if a query pattern presents large errors in some features, the mechanism can still utilize the other more accurate features to find the most probable answers. Finally, this mechanism is especially designed for a datapool with a very large number of patterns, which is frequently encountered in today's applications.

The main problem of the multi-criteria based $k$-pattern recommendation scheme is that it involves a lot of disk I/O's for comparisons. We therefore incorporated the R-tree, which is the most efficient tool in reducing disk I/O the in multi-criteria recommendation field, to accelerate recommendations. The R-tree was initially designed to store planar data in space [8]. However, for years, researchers have been demonstrating that using the R-tree to store and process high-dimensional data is extremely efficient. Thus, a number of R-tree-based algorithms have been designed for high-dimensional space applications such as database queries [22]. In this study, the conventional R-tree cannot be directly applied to the problem, so we devised three novel strategies based on the R-tree to process the target problem, including the forward strategy, the backward strategy, and the hybrid strategy. The forward strategy is the most straightforward method among these strategies. The backward strategy is able to reduce the number of I/O's of the forward strategy based on some theorems. The third strategy is a hybrid strategy, which combines the previous two strategies so that it owns the advantages of both strategies. The simulation result shows that the hybrid strategy requires the least number of I/O's in all cases.

The rest of this paper is organized as follows. The forward and the backward strategy are described in Sections 2 and 3. Section 4 introduces the hybrid strategy. The simulation results are given in Section5. Finally, this paper is concluded in Section 6.

## 2. FORWARD STRATEGY

This section introduces the forward strategy that uses an R-tree to reduce the I/O cost of accessing the datapool in the proposed mechanism. Three data structures, an R-tree, an S_heap, and a linked list, are used in this method. The R-tree is used to record the features of the patterns, which is established prior to executing a search process. The S_heap is used to maintain the temporary information when operating the strategy. The linked list is used to record the result of the mechanism.

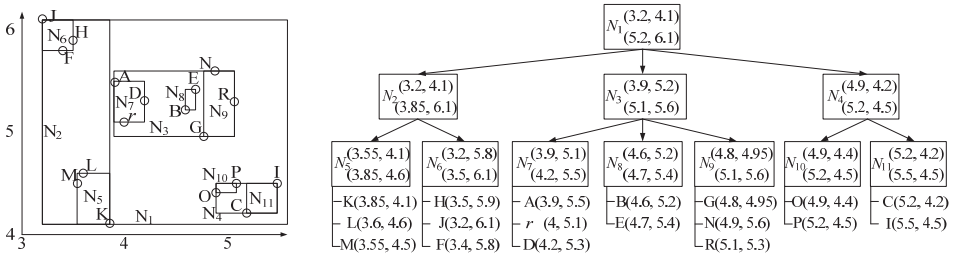
The forward strategy possesses two functions, the mapping function and the search-

ing function. Assume the result of the identification stage is a pattern *r* and the set of patterns in the R-tree is **P**. The mapping function first evaluates the absolute values of the differences between *r* and the patterns in **P**, and then stores the result of the mapping function in the S_heap. The searching function then utilizes S_heap to evaluate the score of each pattern, whose results are then returned to the user. Before we study the forward strategy, the structure of R-tree is briefly introduced in Section 2.1. Then, the strategy is presented in the following subsections.

### 2.1 A Brief Introduction of the R-tree

A pattern can be considered as a point in a hyperspace whose dimensions correspond to the features of the pattern. The main purpose of the R-tree [8] is to organize the patterns (*i.e.*, points) with the similar features (*i.e.*, close to each other) into a node, whose corresponding region bounded by the node is named a minimum bounding rectangle (MBR). We use an example in Fig. 2 to briefly introduce the R-tree. The number of features of a pattern and the capacity of a node in this example are two and three, respectively. Each point in Fig. 2 (a) is a pattern while each rectangle represents the MBR of a node. Fig. 2 (b) is the tree structure of Fig. 2 (a). Two types of nodes are included in this tree, which are the leaf node and the internal node. A leaf node, such as $N_5$, records the features of some patterns. An internal node, such as $N_2$, stores the information of its child MBRs. Both of the leaf and the internal nodes use the coordinates of the bottom-left and the upper-right corner to record the region of MBR, as shown in Fig. 2 (b). Expanding an MBR lets us acquire the patterns and the MBRs (if any) inside this MBR.

Note that this R-tree is normally stored in a secondary storage, which is a much slower device than main memory. Hence, loading a node of an R-tree to main memory is costly comparing to the memory access and the cpu time. A good search strategy should be able to avoid it as much as possible. The detailed mechanism of R-tree, such as insertion and deletion of nodes, can be found in [8].




(a) An index result of the R-tree. (b) The tree structure of the R-tree.

Fig. 2. An example of R-tree.

### 2.2 Mapping Function

Assume the result of the identification stage is pattern *r*. *r* will then be used to search for patterns close to *r*. Three relationships between *r* and an MBR/a point of the R-tree are summarized in Fig. 3. Note that the relationships introduced in the following

can be easily extended to a data space higher than two (*i.e.*, patterns with more than two features). In Fig. 3, $r$ is the origin and for convenience this coordinate system is named the $r$-coordinates. A solid-line rectangle represents an MBR and a black point is a pattern. A white point and a dotted-line rectangle represent respectively the image pattern and the image MBR after mapping with respect to the origin $r$. Note that as the concept of mapping function is quite naive and is not the key point of this work. We only use the following two cases to explain it. Other cases can be easily derived by readers. The first example is the Example 1 in Fig. 3. As the original $p_3$ falls in the third quadrant of the $r$-coordinates, it must be mapped to the first quadrant of the $r$-coordinates. To achieve this objective, $p_3$ must be mapped with regard to $r$. Originally at (−1, −1), $p_3$ moves to (1, 1) after mapping. The second example is the Example 4 in Fig. 3. As the original MBR falls in the third quadrant of the $r$-coordinates, it must be mapped to the first quadrant of the $r$-coordinates. To achieve this objective, the lower left corner (*i.e.*, $p_1$) and upper right corner (*i.e.*, $p_2$) of the MBR must be mapped with regard to $r$. Assuming that the original $p_1$ and $p_2$ were located at (−3, −3) and (−1, −1), respectively, the new $p_1$ and $p_2$ will fall on (3, 3) and (1, 1) after mapping. Thus, the location of the image MBR will become the MBR with (1, 1) and (3, 3) as the lower left and upper right corners.

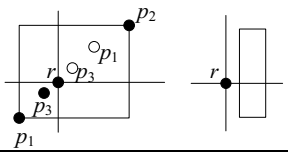
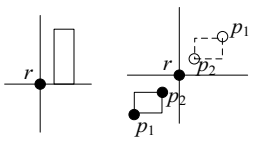
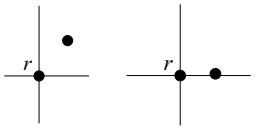
| Type | MBR occupying more than two quadrants in $r$-coordinates | | MBR occupying only one quadrant in $r$-coordinates | | Point in $r$-coordinates | |
|---|---|---|---|---|---|---|
| | Example 1 | Example 2 | Example 3 | Example 4 | Example 5 | Example 6 |
| Examples |  | |  | |  | |
| Action | Expand the MBR | | None | | None | |

Fig. 3. Spatial relationships between $r$-coordinates and MBR/point of an R-tree.

After the above mapping, the image MBRs and the image patterns are inserted into an S_heap. In the S_heap, they are sorted in ascending order according to the summation of their coordinates. For an MBR, the summation is made on the coordinates of the bottom-left corner of this MBR. Hereafter, we simply name it the summation of an MBR (or a point) for brevity. An interesting fact about this summation is that the pattern with a greater summation can never be better than the one with a smaller summation. This means if the patterns and MBRs in the S_heap are sorted according to their summations and if $p$ is not an answer, then all patterns and MBRs behind pattern $p$ in the S_heap need not be considered as they cannot be better than $p$. The indication of this property is that computation and I/O can be reduced, so the performance can be enhanced.

### 2.3 Searching Function

Patterns and MBRs in the S_heap are now retrieved, on which the searching function is performed. The searching function involves a series of simple operations, as shown in Fig. 4. For ease of presentation, both a pattern and an MBR in the S_heap are named an *element* hereafter.

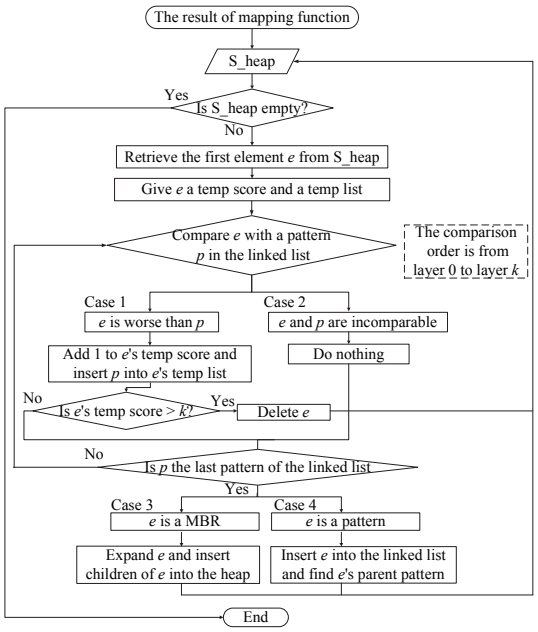
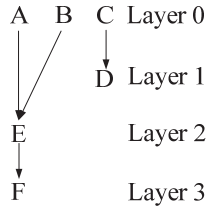
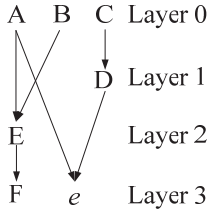



Fig. 4. The flow chart of the searching function of the forward strategy.




(a) A linked list.




(b) Insert *e* into the linked list.

Fig. 5. The examples of a linked list.

The function retrieves the first element of the S_heap and assigns a temp score and a temp list to this element. The temp score is used to record the score of the element while the temp list is used to record the patterns that are better than the retrieved element. In the beginning, the temp score is set to 0 and the temp list is empty. Next, when the first element, say *e*, of the S_heap is retrieved, it is compared with the patterns in a linked list (referring to the process in Fig. 4). In the beginning, the linked list is empty. But for the sake of explaining how the forward strategy works, let us assume that the linked list has a content as shown in Fig. 5 (a). The comparison order is from layer 0 to layer 3 and within each layer from left to right. In Fig. 5 (a), for example, the retrieved element *e* will first be compared with A, then B and C, and then D, E, and F.

When *e* is compared with a pattern in the linked list, the comparison is based on the concept given in the following definition.

**Definition 1:** Given a datapool with $d$ features, an identification result $r$, and two patterns $p_1$ and $p_2$. The absolute difference of $p_1$ and $r$ is $(v_{11}, v_{12}, \ldots, v_{1d})$ and that of $p_2$ and $r$ is $(v_{21}, v_{22}, \ldots, v_{2d})$. We say that $p_1$ is worse than $p_2$ if $v_{1i} > v_{2i}$ for all $1 \le i \le d$. If $v_{1i} > v_{2i}$ is only true for some (but not all) $i$, then we say that they are incomparable. ❑

The condition $v_{1i} > v_{2i}$ for all $1 \le i \le d$ in this definition implies that $p_1$ is not closer to $r$ than $p_2$ in all features. In this case, the possibility that $p_1$ becomes the answer of the query pattern is smaller than that of $p_2$. If however the condition in Definition 1 does not meet (*i.e.*, $p_1$ is closer to $r$ than $p_2$ in some features and $p_2$ is closer to $r$ than $p_1$ in the other features), then both $p_1$ and $p_2$ have a chance to be the answer of the query pattern. For these two cases, we do the following.

Let *e* be the element retrieved from S_heap and *p* be a pattern of the linked list.

**Case 1: *e* is worse than *p*.** In this case, we add one to the temp score of *e* and insert *p* into the temp list of *e*. If the temp score of *e* is greater than *k* after the addition, then *e* will be deleted from the S_heap and the process moves on to the next step. This is because if *e* is a pattern, then the final score of this pattern must be greater than *k*. If *e* is an MBR, then the final scores of all patterns in this MBR should be greater than *k* as well. Note that *e* can never be better than *p*, as the summation of *p* must be smaller than *e*.

**Case 2: *e* and *p* are incomparable.** In this case, no further operation is needed.

Note that during the comparison, patterns in the linked list are not removed. They are only used to determine whether *e* can be deleted from the S_heap or, if not, how to deal with *e* for further processing. After the processing of case 1 and case 2, the searching function checks whether *p* is the last pattern in the linked list in this series of comparisons. If so, the searching function proceeds to the process of case 3 and 4. Otherwise, *e* is compared with the next pattern in the linked list.

**Case 3: *p* is the last pattern in the linked list, the temp score of *e* is not greater than *k*, and *e* is an MBR.** In this case, the score of the patterns in this MBR may be smaller than or equal to *k*. Hence, this MBR needs to be expanded and all child MBRs or patterns within this MBR should be inserted to the S_heap. Note that before the insertion, the image coordinates of these child MBRs and patterns are obtained by using the $(I_x, I_y)$ of the MBR and the coordinates of *r*. The summations of the image coordinates are then used to determine the order of these child MBRs and patterns in the S_heap.

**Case 4: *p* is the last pattern in the linked list, the temp score of *e* is not greater than *k*, and *e* is a pattern.** In this case, this pattern *e* is an answer of the search process. So, *e* is inserted into layer *i* of the linked list where *i* is the score of *e*. That is, all patterns with a score smaller than *k* (*i.e.*, they are the answer of the query) and the relationships between these patterns are recorded in a linked list. An example of a four layer linked list with $k$=3 is shown in Fig. 5 (a). A pattern with score *i* is stored in layer *i*. Each layer may have multiple patterns, in which the order of these patterns (of the same layer) is arranged according to the order that they are inserted into the linked list. The arrows in the linked list record the relationships between the patterns. For example, Fig. 5 (a) shows that the parent pattern of F is E and the parent patterns of E are A and B. Hence, F is worse than A, B, and E. They are however incomparable to C and D. Assume that the temp list of *e* includes patterns A, C, and D. Then we know *e* is worse than A, C, and D. Also, we know from Fig. 5 (a) that C is better than D. Hence, C cannot be the parent pattern of *e*, and therefore A and D are the parent patterns of *e*, as shown in Fig. 5 (b).

## 3. BACKWARD STRATEGY

In this section, we propose a new strategy, named the backward strategy, to improve the efficiency of the forward strategy. The main advantage of the forward strategy is using an R-tree to reduce the I/O cost of accessing the datapool. It however cannot avoid

the need of comparing the retrieved element with all patterns in the linked list until it has found $k$+1 patterns better than it. Hence, it requires some I/O's for accessing cost of accessing the linked list. The cost can be significant when $k$ is large.

The backward strategy is designed to remedy the flaw of the forward strategy. Hence, the flow chart of the backward strategy is similar to that of the forward strategy. The major difference is that when comparing the element retrieved from the S_heap with the patterns in the linked list, the comparing order is opposite to that in the forward strategy. Such a change help the algorithm to reduce the number of comparisons in the searching function. The mapping function of the backward strategy is the same as that in the forward strategy. Hence, we only introduce the searching function of the backward strategy in the following. Before we explain this searching function, we first introduce a series of lemmas and theorems that will be used in the backward strategy.

### 3.1 Theoretical Foundation of the Backward Strategy

We first introduce the transitivity of relationships of the patterns and the elements. Given two patterns $p_1$ and $p_2$ and an element $e$ (either a pattern or an MBR), if $p_2$ is worse than $p_1$ and $e$ is worse than $p_2$, then it is quite clear that $e$ must be worse than $p_1$. According to this fact, we are able to derive Theorem 1 in the following.

**Theorem 1:** Given an element $e$ and a pattern $p$ with a score $m$, if $e$ is worse than $p$, then the score of $e$ must be equal to or greater than $m$+1.

***Proof*:** First, the score of $p$ being $m$ implies that $p$ is worse than another $m$ patterns, say $p'_1, p'_2, \ldots, p'_m$. Hence, if $e$ is worse than $p$, then $e$ should be worse than not only $p$, but also $p'_1, p'_2, \ldots, p'_m$. That is to say, the score of $e$ must be equal to or greater than $m$+1. ❑

Theorem 1 ensures that if an element is worse than a pattern of layer $k$ in the linked list, then it can be deleted from the S_heap.

**Lemma 1:** Given an element $e$ and $n$ patterns $p_1, p_2, \ldots, p_n$, if (1) the score of $p_1$ is $m$; (2) $p_2, p_3, \ldots, p_n$ are not better than $p_1$; and (3) $e$ is worse than $p_1, p_2, \ldots, p_n$, then the score of $e$ must be equal to or greater than $m$+$n$. ❑

**Lemma 2:** Given two patterns $p_1$ and $p_2$, if the scores of $p_1$ and $p_2$ are the same, then $p_1$ and $p_2$ must be incomparable. ❑

From these two lemmas, we can derive Theorem 2.

**Theorem 2:** Given $n$ patterns $p_1, p_2, \ldots, p_n$ in layer $m$ of the linked list, if an element $e$ is worse than $p_1, p_2, \ldots, p_n$, then the score of $e$ must be equal to or greater than $m$+$n$.

***Proof*:** According to Lemma 2, $p_1, p_2, \ldots, p_n$ being in the same layer implies that these patterns are incomparable. Hence, from Lemma 1, the score of $e$ must be equal to or greater than $m$+$n$, as the score of $p_1$ is $m$ and $p_2, p_3, \ldots, p_n$ are not better than $p_1$. ❑

Theorem 2 can be further extended Theorem 3.

**Theorem 3:** If an element $e$ is worse than $n_m$ patterns of layer $m$ of the linked list, $n_{m+1}$

patterns of layer $m$+1, …, $n_{m+(k\text{-}1\text{-}m)}$ patterns of layer $m$+($k$–1–$m$), then the score of $e$ must be equal to or greater than $m+n_m+\sum_{i=1}^{k-1-m} n_{m+i}$. ❑

Finally, we extend Theorem 3 to Theorem 4.

**Theorem 4:** Assume an element $e$ is worse than $n_m$ patterns of layer $m$, $n_{m+1}$ patterns of layer $m$+1, …, $n_{m+(k\text{-}1\text{-}m)}$ patterns of layer $m$+($k$–1–$m$), in which each of these patterns $p_k$ ($1< k < n_m+n_{m+1}+...+n_{m+(k\text{-}1\text{-}m)}$) is associated with the number of patterns, $u_{kj}$, that is better than $p_k$ in layer $j$ ($0 < j < m-1$). Then the score of $e$ must be equal to or greater than $n_m+\sum_{i=1}^{k-m} n_{m+i}+\sum_{j=0}^{m-1}\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$.

***Proof*:** First, the assumption of this theorem indicates that $e$ is worse than at least $n_m+\sum_{i=1}^{k-m} n_{m+i}$ patterns of layer $k$–1 to $m$. Next, if $e$ is worse than $p_1$, $p_2$, …, and $p_k$, then in layer $j$, $e$ must be worse than at least $\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$ patterns. This is because $\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$ is the minimum number of patterns that are better than $p_1$, $p_2$, …, or $p_k$ in layer $j$. Hence, according to the transitivity relationship between the patterns, $e$ must be worse than at least $\sum_{j=0}^{m-1}\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$ patterns of layer 0 to layer $m$–1. From the above two arguments, we know that $e$ is worse than at least $n_m+\sum_{i=1}^{k-m} n_{m+i}+\sum_{j=0}^{m-1}\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$ patterns of layer 0 to layer $k$–1 and the score of $e$ must be equal to or greater than $n_m+\sum_{i=1}^{k-m} n_{m+i}+\sum_{j=0}^{m-1}\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-1-m)}}(u_{kj})$. ❑

For an element $e$ compared with the patterns of layer $k$ back to layer $m$ in the linked list where $k > m$, Theorem 3 is able to examine whether the score of $e$ is greater than $k$ by using the patterns of layer $k$–1 to layer $m$ while Theorem 4 is able to do the same task by using the patterns of layer $k$–1 to layer 0.

### 3.2 The Searching Function of the Nackward Strategy

The flow chart of the searching function of the backward strategy (Fig. 6) is similar to that of the forward strategy (Fig. 4). Differences are marked in boldface in Fig. 6. We start the introduction of the proposed searching function from its linked list, as it is the major part that is different from the forward strategy. Each pattern in the linked list of the backward strategy has an additional B_array. If the threshold of score is $k$, then the B_array is a 1-by-$k$ array. An example of B_array is shown in Fig. 7, in which $k$=3. The B_array [2, 0, 1] of F in this figure implies that F is worse than two patterns of layer 0 (*i.e.*, A and B) and one pattern of layer 2 (*i.e.*, E).

We next introduce the process of this function. The first element of the S_heap is retrieved and assigned a temp score, a temp list and a max_array. The temp score records the number of patterns better than $e$. Assume that the comparison has been done on layer $k$ up to layer $m$. Then, the temp score of $e$ at this time is equal to $n_m+\sum_{i=1}^{k-m} n_{m+i}$ according to Theorem 4. The temp list stores the patterns that are better than $e$. This list is used to find the parent patterns of $e$ and establish the B_array of $e$ when $e$ needs to be inserted into the linked list. The max_array is a 1-by-$k$ array, which is used to evaluate $\sum_{j=0}^{m-1}\max_{k=1}^{n_m+n_{m+1}+...+n_{m+(k-m)}}(u_{kj})$ mentioned in Theorem 4. Assume that the patterns better than $e$ in the linked list are $p_1$, $p_2$, …, $p_n$, and the B_array of $p_i$ is [$v_{i0}$, $v_{i1}$, …,$v_{ik\text{-}1}$]. Then the max_array of $e$ is [max($v_{10}$, $v_{20}$, …,$v_{n0}$), max($v_{11}$, $v_{21}$, …,$v_{n1}$), …, max($v_{1k\text{-}1}$, $v_{2k\text{-}1}$, …,

$v_{nk\text{-}1})$]. The value of the $\sum_{j=0}^{m-1} \max_{k=1}^{t_m+t_{m+1}+\ldots+t_{m+(k-m)}}(u_{kj})$ is eqaul to the summation of the first $m$ values (*i.e.*, layer 0 to layer $m-1$) of the max_array.

Let $e$ be the element retrieved from S_heap and $p$ be a pattern of the linked list.

**Case 1: *e* is worse than *p*.** This case is partly different from the case 1 of the forward strategy. In this case, we first add one to the temp score of $e$ and insert $p$ into the temp list of $e$. Next, if $p$ is from layer $k$, then according to Theorem 1 the final score of $e$ must be greater than $k$. Hence, $e$ is deleted from the S_heap and the process moves on to the next step. If $p$ is from layer $m$ instead of layer $k$, then the max_array of $e$ is updated by the B_array of $p$. The updated max_array is then used to examine whether the condition of Theorem 4 is met. If so, the score of $e$ must be greater than $k$ and thus $e$ can be deleted from the S_heap. Otherwise, $e$ is compared with the next pattern in the linked list. The whole process is also clearly shown in Fig. 6.

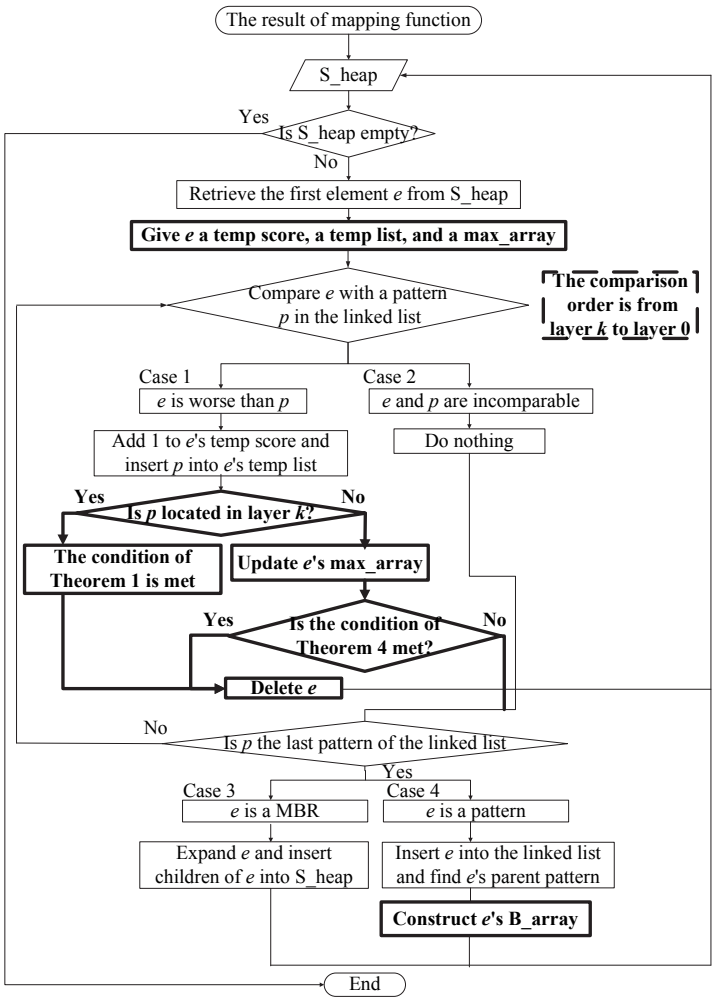



Fig. 6. The flow chart of the searching function of the backward strategy.

**Case 2: *e* and *p* are incomparable.** No further operation is needed, same as the case 2 of the forward strategy.

If *p* is not the last pattern in the linked list, then *e* will be compared with the next pattern in the linked list. Otherwise, we move on to case 3 and case 4.

**Case 3: *p* is the last pattern in the linked list, the temp score of *e* is not greater than *k*, and *e* is an MBR.** The process of this case is the same as that of the forward strategy. As the score of the child elements within the MBR may be smaller than or equal to *k*, this MBR needs to be expanded. Its child elements are then inserted into the S_heap in ascending order according to the summation of their image coordinates.

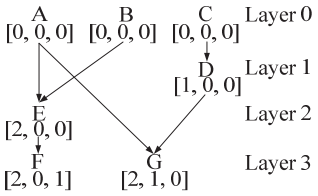



Fig. 7. The example of the linked list in the backward strategy.

**Case 4: *p* is the last pattern in the linked list, the temp score of *e* is not greater than *k*, and *e* is a pattern.** This case is partly different from the case 4 of the forward strategy. This *e* is an answer of the proposed mechanism and thus inserted into the linked list. The difference is that an additional B_array for *e* is established. The B_array is obtained from the temp list of *e*. For example, if the temp list of F in Fig. 7 is [A, B, E], then the B_ array of F is [2, 0, 1]. This is because F is worse than two patterns (*i.e.*, A and B) of layer 0, a pattern (*i.e.*, E) of layer 2, and not worse than any patterns of layer 1.

## 4. THE HYBRID STRATEGY

The hybrid strategy, as indicated by its name, is a combination of the forward strategy and the backward strategy. Hence, it also possesses the mapping function and the searching function. The main design purpose is to further reduce the cost for accessing patterns during the processing. The mapping part is the same as the previous two strategies. Hence, we also introduce only the searching part in this section.

The basic idea of the hybrid strategy is that the comparing order of an element, say *e*, in the S_heap with a pattern, say *p*, in the linked list is mixed. It starts by following the backward strategy to compare *e* with the patterns in the last layer (*i.e.*, layer *k*) of the linked list. When the searching function finds that the access cost of this backward comparing order would become greater than comparing in a reverse order (*i.e.*, following the forward strategy to compare from layer 0), the search process begins to execute the forward strategy. The major challenge of the hybrid strategy is to find the layer *i* ($0 < i < k$), *i.e.*, the turning point, at which the search process reverses the comparing order. This involves a cost determination process. We introduce this process in the following.

Let *e* be the element to be dealt with (*i.e.*, retrieved from the S_heap) and *M* be its parent MBR. As the searching process always starts from the top levels of the R-tree to

the bottom levels, the parent of $e$ (that is, $M$) must have been processed (*i.e.*, compared with the linked list) before $e$ is processed. Hence, the max_array of $M$ is known when $e$ is going to be processed. Information in this max_array of $M$ is useful during $e$'s comparisons, which will be clear later. We have the following lemma regarding to the max_arrays of $M$ and $e$.

**Lemma 3:** Given an element $e$ and its parent *MBR* $M$, if the *max_array* of $e$ is $[ne_0, ne_1, \ldots, ne_{k-1}]$ and the *max_array* of $M$ is $[nm_0, nm_1, \ldots, nm_{k-1}]$, then $ne_i$ should be greater than or equal to $nm_i$, where $0 \le i \le k$. ❑

This is because the bottom-left corner of $M$, if there is a pattern at this corner, is definitely better than $e$ (because $M$ contains $e$). Thus, all patterns better than $M$ must be better than $e$ too. From this, we can safely inference that the number of patterns better than $M$ recorded in the max_array of $M$ can be known as the least number of patterns better than $e$. For this reason, we say that the max_array of $M$ is also the "min_max_array" of $e$, where the min_max_array stands for the *minimum* max_array.

As we said, the comparison starts from the bottom layer (*i.e.*, layer $k$) of the linked list. When the comparison proceeds up to layer $i$ ($0 < i < k$) and enough patterns better than $e$ are found, then $e$ can be deleted without further processing. In Theorem 5 we give this number.

**Theorem 5:** When comparing element $e$ with the patterns of layer $i$, if the *max_array* of $e$ is $[ne_0, ne_1, \ldots, ne_{k-1}]$ at this moment, then $e$ can be deleted from the *S_heap* if at least $k+1-\max(i, \sum_{m=0}^{i-1} ne_m) - ne_i - \sum_{m=i+1}^{k-1} ne_m$ patterns better than $e$ are found in layer $i$.

***Proof*:** If $i \ge \sum_{m=0}^{i-1} ne_m$, the equation in this theorem is reduced to $k+1-i-ne_i-\sum_{m=i+1}^{k-1} ne_m$. The values $ne_i, ne_{i+1}, \ldots, ne_{k-1}$ in the max_array of $e$ indicate that $e$ is worse than $ne_i$ patterns of layer $i$, $ne_{i+1}$ patterns of layer $i+1$, …, and $ne_{k-1}$ patterns of layer $k-1$. Theorem 3 mentioned that if the above argument is true, then $e$ is worse than at least $i+ne_i+\sum_{m=i+1}^{k-1} ne_m$ patterns. That is to say, if additional $k+1-i-ne_i-\sum_{m=i+1}^{k-1} ne_m$ patterns better than $e$ in layer $i$ are found, then $e$ can be deleted from the S_heap (*i.e.*, $e$ is worse than at least $k+1$ patterns).

On the other hand, if $i < \sum_{m=0}^{i-1} ne_m$ , then the expression becomes $k+1-\sum_{m=0}^{i-1} ne_m - ne_i - \sum_{m=i+1}^{k-1} ne_m = k+1-\sum_{m=0}^{k-1} ne_m$ . As the max_array of $e$ is $[ne_0, ne_1, \ldots, ne_{k-1}]$, $e$ is worse than at least $\sum_{m=0}^{k-1} ne_m$ patterns in the linked list. Hence, if additional $k+1-\sum_{m=0}^{k-1} n_m$ patterns better than $e$ in layer $i$ are found, $e$ can be deleted from the S_heap. ❑

As this number (of patterns better than $e$ in order to delete $e$) can be evaluated beforehand, we can estimate the probability that $e$ is able to be deleted when compared with patterns in layer $i$. This probability is given in Theorem 6. But before that a lemma for calculating approximate probability is introduced.

**Lemma 4:** Given a datapool with $d$ features, an element $e$, and $n$ patterns $p_1(v_{11}, v_{12}, \ldots, v_{1d})$, $p_2(v_{21}, v_{22}, \ldots, v_{2d})$, …, and $p_n(v_{n1}, v_{n2}, \ldots, v_{nd})$. Let all features of the patterns be normalized to the range [0, 1], *i.e.*, $0 \le v_{ij} \le 1$ for all $1 \le i \le n$ and $1 \le j \le d$. The probability that $e$ is worse than these $n$ patterns is $\Pi_{i=1}^{d}\left(1-\max(v_{1i}, v_{2i}, \ldots, v_{ni})\right)$.

***Proof*:** For ease of comprehension, we prove this lemma in a two dimensional space (as shown in Fig. 8). It can be easily extended to a higher dimensional case. The minimum and the maximum value of the features in this figure are 0 and 1, respectively. Patterns in region $\alpha$ are worse than $p_1(v_{11}, v_{12})$, $p_2(v_{21}, v_{22})$, and $p_3(v_{31}, v_{32})$. As the area of the entire area is 1, the area of region $\alpha$ is $\Pi_{i=1}^{2}(1-\max(v_{1i}, v_{2i}, v_{3i}))$. Therefore, the probability that $e$ is located in region $\alpha$ and worse than $p_1$, $p_2$, …, and $p_n$ is $\Pi_{i=1}^{d}(1-\max(v_{1i}, v_{2i}, ..., v_{ni}))$. ❑

**Theorem 6:** Given a datapool with $d$ features, an element $e$, and $n$ pattern $p_1(v_{11}, v_{12}, \ldots, v_{1d})$, $p_2(v_{21}, v_{22}, \ldots, v_{2d})$, …, $p_n(v_{n1}, v_{n2}, \ldots, v_{nd})$ in layer $i$ of the linked list. All features of the patterns are normalized to the range [0, 1]. The number of patterns in layer $i$ of the linked list is $nl_i$. If there are $nd_i$ patterns in layer $i$ better than $e$ so that $e$ can be deleted, then the probability $pd_i$ of deleting $e$ from the *S_heap* in layer $i$ is

$$\begin{cases} 0 & , \text{ if } nd_i > nl_i \\ \Pi_{i=1}^{d}(1-\max(v_{1i}, v_{2i}, ..., v_{ni})), & \text{if } nd_i \le nl_i \end{cases}. \tag{1}$$

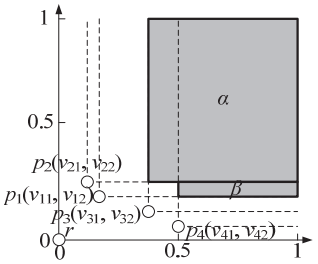



Fig. 8. An example of Lemma 4 and Theorem 6.

***Proof*:** We first consider the case that $nd_i > nl_i$. As $e$ can never be deleted in this case, $pd_i$ of $e$ is 0. Next, we consider the case $nd_i \le nl_i$. Let us also refer to Fig. 8. The values of $d$ and $nd_i$ in this figure are 2 and 3, respectively. $p_1(v_{11}, v_{12})$, $p_2(v_{21}, v_{22})$, $p_3(v_{31}, v_{32})$, and $p_4(v_{41}, v_{42})$ are all located in layer $i$ of the linked list. The order of these patterns of layer $i$ is $p_1$, $p_2$, $p_3$, and $p_4$. Hence, $e$ should be first compared with $p_1$, $p_2$, $p_3$, and then $p_4$. If $e$ can be deleted after comparing $p_1$, $p_2$, and $p_3$, then $e$ must be located in region $\alpha$. Similarly, if $e$ is deleted after comparing $p_1$, $p_2$, $p_3$, and $p_4$, then $e$ may be located in region $\alpha$ or in region $\beta$. However, the area of region $\beta$ is usually much smaller than that of region $\alpha$. The area of region $\alpha$ in Fig. 8 is $(1-v_{31})(1-v_{22})$ and that of $\beta$ is $(1-v_{41})(v_{22-12})$. $(1-v_{31})$ is greater than $(1-v_{41})$. $(1-v_{22})$ is usually much greater than $(v_{22}-v_{12})$. As the entire area is 1, $pd_i$ of $e$ is therefore approximately the area of region $\alpha$, which is $\Pi_{i=1}^{d}(1-\max(v_{1i}, v_{2i}, ..., v_{ni}))$. ❑

This theorem implies that the approximate $pd_i$ of $e$ is only affected by the coordinates of the first $nd_i$ patterns of layer $i$. As $v_{ij}$ in the linked list are all known, this probability can be calculated before $e$ is actually compared with the patterns of layer $i$. That is to say, we should be able to find the proper layer $i$ to reverse the comparing order, *i.e.*, changing from backward to forward checking, before $e$ is compared with any pattern of the linked list.

Let the element under comparison be $e$, the numbers of patterns of layer 0 to layer $k$

be respectively $nl_0$, $nl_1$, …, $nl_k$, and the probabilities of deleting $e$ in layer 0 to $k$ be respectively $pd_0$, $pd_1$, …, $pd_k$. We start the comparison from pattern in layer $k$ of the linked list. Let us assume that we perform the comparison upward till layer $x$ ($x < k$). That is, patterns in layer $k$ upward to layer $x$ have all been compared with $e$. Assume that this is the turning point and now we reverse the comparing order downward from layer 0 to layer $x$–1. Then, the expected cost for accessing patterns from layer 0 to layer $x$–1 can be calculated in the following manner. The probability that $e$ can be deleted from S_heap when it is compared with the patterns of layer $j$ (*i.e.*, $e$ has not been deleted from S_heap after comparing with patters of layer 0 to layer $j$–1) is $\left(\Pi_{i=0}^{j-1}(1-pd_i)\right)pd_j$. The cost for accessing patterns of layer 0 to layer $j$ for comparison is $\sum_{i=0}^{j} nl_i$. Therefore, the expected cost for accessing patterns of layer 0 to layer $x$–1 is

$$\sum_{j=0}^{k-x}\left(\left(\Pi_{i=0}^{j-1}(1-pd_i)\right)pd_j\sum_{i=0}^{j}nl_i\right). \tag{2}$$

If, on the other hand, the turning point is not at layer $x$ (as in the above) but at layer $x$–1, then the patterns of layer $x$–1 would be accessed and compared with $e$ and then we reverse the pattern comparison from layer 0 down to layer $x$–2. The total access cost for this case is therefore the cost of accessing patterns in layer $x$–1 plus the cost of accessing patterns from layer 0 to layer $x$–2. $pd_{k-x}nl_{k-x}$ is the expected cost of accessing patterns of layer $x$–1 to compare with $e$ and finding that $e$ can be deleted from the S_heap during the comparison. As for the access cost for patterns of the above layers (*i.e.*, layer 0 to layer $x$–2), it can be calculated in the following manner. The probability that $e$ can be deleted from the S_heap when it is compared with the patterns of layer $j$ is $(1-pd_{k-x})\left(\Pi_{i=0}^{j-1}(1-pd_i)\right)pd_j$. The corresponding access cost for patterns of layer $j$ is $(nl_{k-x}+\sum_{i=0}^{j}nl_i)$. Hence, we have the expected cost of accessing patterns of layer $x$–1 first and then turning to layer 0 down to layer $x$–2.

$$pd_{k-x}nl_{k-x}+\sum_{j=0}^{k-x-1}\left((1-pd_{k-x})\left(\Pi_{i=0}^{j-1}(1-pd_i)\right)\cdot pd_j\left(nl_{k-x}+\sum_{i=0}^{j}nl_i\right)\right) \tag{3}$$

Note that in the above two cases, the cost of accessing patterns of layer $k$ upward to layer $x$ is not included. It is because this cost in both cases is the same. Hence, there is no need to include that part. Cost expressions (2) and (3) can be used to determine the turning point from the backward strategy to the forward strategy. Let us assume that we have finished the comparisons for layer $k$ up to layer $x$. We will calculate the costs of expres sion (2) and expression (3). If cost(2) ≥ cost(3), then we continue the backward strategy (to access patterns of layer $x$–1). Otherwise, we change the execution to the forward strategy (to access patterns of layer 0). As this strategy is a combination of the previous two and not difficult to comprehend, we omit its flow chart.

**Table 2. A summary of experiment parameters.**

| Parameter | Values |
|---|---|
| Number of the features in the datapool, $d$ | 2, 3, **4**, 5, 6 |
| Number of the error-features in the datapool, $m$ | 1, 2, 3 |
| The upper bound of the percentage of the error in the reference-features, $x$ | 0.1% to 11% |
| Threshold of the score of the datapool, $k$ | 1, 10, 25, **50**, 75, 100 |
| Number of patterns in the datapool | 1M |

## 5. SIMULATION

A set of simulations on face identification was conducted in this section to demonstrate the effectiveness and the efficiency of the recommendation system. The datapool used in this simulation is a synthetic face feature datapool. This datapool contains 1,000,000 patterns and each pattern has six commonly used features. They are the height of the face, the height of the forehead, the widths of the left eye and the right eye, the width of the nose, and the width of the mouth [12, 17]. Note that we use the synthetic feature datapool in this simulation instead of a real pattern datapool for the following two reasons. First, the main focus of this paper is not to extract the features from the patterns but to provide a method for quickly finding the similar patterns from the datapool. Using synthetic datapool allows us to avoid the need of extracting the features from the real pattern datapool. Second, for the purpose of testifying the feasibility of the recommendation system, we need to have a datapool of a large number of patterns. However, most of the existing face datapools, such as FERET [18], YaleB [6], and LFW [16] have only about ten thousand patterns, which is quite small for our purpose. Hence, a synthetic datapool is used in the simulation.

Two parts are included in this simulation. The first part is to study the accuracy of the proposed recommendation system. The second part is to compare the efficiency of the brute force strategy and the three strategies devised in this paper. The parameters and their varying ranges are summarized in Table 2, in which default values are marked in boldface. These parameters includes (1) the number of features $d$ in the datapool, which varies from two to six and the default value is four; (2) the threshold $k$ of the score, which varies from 1 to 100 and the default value is 50; and (3) the number of patterns in the datapool, which is fixed at 1M. Each performance curve shown in the figures represents an average of the experimental results of 30 datapools. All of the experiments were performed on an Intel i7-3770 CPU at 3. 40GHz with 4GB main memory, running on Microsoft Windows XP. All the programs were written in MATLAB®.

### 5.1 The Accuracy of the Recommendation System

Assume that the query pattern is $q$, the correct identification answer for $q$ is $c$, and the answer set returned by the proposed recommendation system is $L$. Normalized Discounted Cumulative Gain (NDCG) can be used to demonstrate the performance of the proposed algorithms. From a random selection of 200 query patterns, we determined whether the answer set returned by the proposed recommendation system contains the correct identification answer for $q$. Fig. 9 is the NDCG pertaining to $q$ under various $d$, $k$, and the error of $q$. The error of $q$ in this figure is determined by two parameters, $m$ and $x$. $m$ refers to the number of features of $q$ that are significantly different from $c$, where $m < d$. $x$ refers to the maximum difference (as a percentage) of the remaining ($d$–$m$) features between $q$ and $c$. For the ease of presentation, $m$ features with large errors are hereby referred to as *error-features,* whereas the other ($d$–$m$) features are referred to as *reference-features*. Each subfigure in Fig. 9 differs in its combination of $d$ and $m$. Each curve in the subfigure indicates the effect of $k$ on the NDCG of identifying $q$ under the same error conditions.

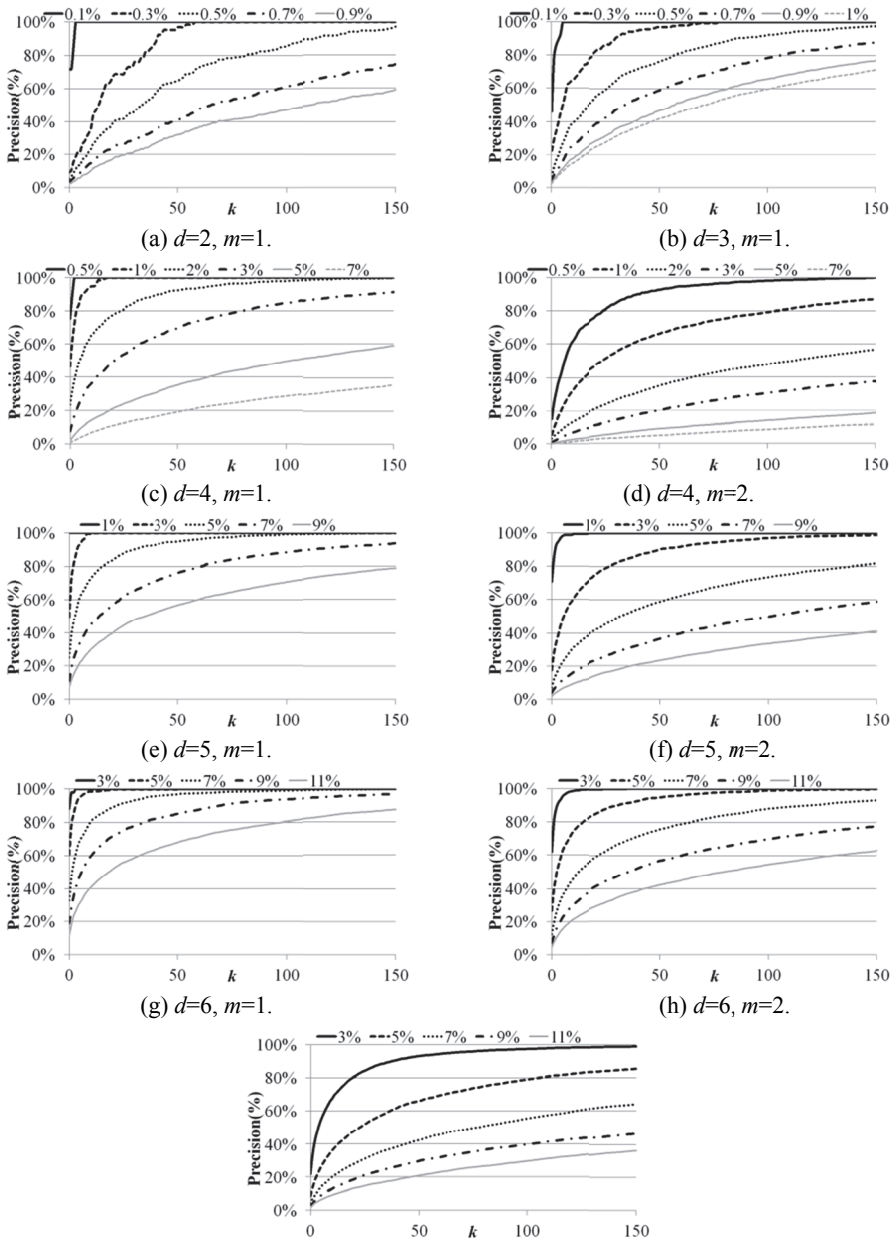



Fig. 9. The accuracy of $q$ by varying $d$, $m$, $x$, and $k$.

As shown in Fig. 9, when $d$, $m$, and $x$ are constant, the NDCG of identifying $q$ increases with $k$. This means that if the error of $q$ is high, then the NDCG of identifying $q$ can be upgraded by increasing $k$ in the proposed recommendation system. We suggest setting $k$ to a value of 50 or greater, which should be sufficient for NDCG to reach a rela-

tively stable state.

When $m$ and $k$ are constants, a higher value for $d$ allows for greater fluctuations in the value of $x$ (*i.e.*, the reference-features). For example, when $d$ is 2 and $m$ is 1 (Fig. 9 (a)), the only curves that attain 100% NDCG are those with $x$ of 0.3% or lower. Increasing $d$ to 6 while maintaining $m$ at 1 (Fig. 9 (g)) makes it possible for all curves with $x$ of 9% or lower to attain 100% NDCG. That is to say, when dealing with a datapool with two features, the recommendation system is able to tolerate only 0.3% error in the reference-features. Nonetheless, the tolerance can be increased to 9% in cases where the patterns of the datapool have six features. This provides an empirical explanation for our assertion that a greater number of features could be expected to produce answers of higher accuracy.

Finally, when $d$, $x$, and $k$ remain constant, NDCG decreases with an increase in $m$. For example, when $d$ is 4 and $m$ is 1, (Fig. 9 (c)), the NDCG when $x$ = 2% is approximately 93% at $k$ = 50. However, keeping $d$ at 4 but increasing $m$ to 2 (Fig. 9 (d)) reduces NDCG to approximately 35% with the same $k$ and $x$. This can be explained by the fact that increasing the value of $m$ (*i.e.*, the number of error-feature increases) decreases the amount of the correct information that can be obtained by the recommendation system from a given query pattern, thereby reducing the NDCG of identifying $q$. Nevertheless, the NDCG remains high when $d$ is 5 or 6. For instance, when $d$ is 5 and $m$ is 2 (Fig. 9 (f)), the NDCG at $k$ = 50 reaches 90% for $x$ = 3%. When $d$ is 6 and $m$ is 3 (Fig. 9 (i)), the NDCG at $k$ = 50 is approximately 67% for $x$ = 5%. This explains why the proposed method is suitable for datapools with a larger number of features.

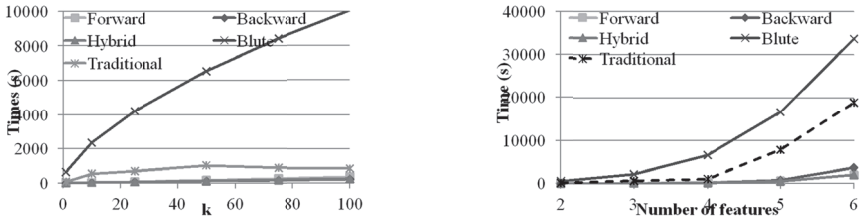
### 5.2 The Performance of the Proposed Strategies

In this subsection, we study the performance of the three proposed strategies. They will be compared with a brute force strategy to see how much improvement can be made.

#### 5.2.1 Comparing the performance of the brute force strategy, traditional identification algorithm and the three proposed strategies

In this section, we provide a performance comparison of the brute force approach, the traditional identification algorithm, and the three proposed methods. The brute force strategy represents a straightforward approach to the implementation of a recommendation system, with no supporting data structure, such as the R-tree or a linked list. Each pattern $p$ in the datapool is compared with all other patterns in the same datapool. If $k$+1 patterns are found to be better than $p$, then the score of $p$ is greater than $k$, thereby indicating that $p$ is not a viable answer for the recommendation system. Only $p$ scores that are smaller than $k$ are returned to the user. Furthermore, the process must continue until all patterns in the datapool have been processed. The traditional identification algorithm searches for one suitable answer at a time using a nearest neighbor query with support from an index structure. The $i$th time that processing is conducted produces the $i$th nearest result for the query pattern. Furthermore, the algorithm is repeated until all of the results are found.

Fig. 10 presents the time costs of each strategy. Fig. 10 (a) was obtained by varying $k$ and Fig. 10 (b) was obtained by varying $d$. As expected, the time costs of the brute




(a) By varying *k*. (b) By varying *d*.

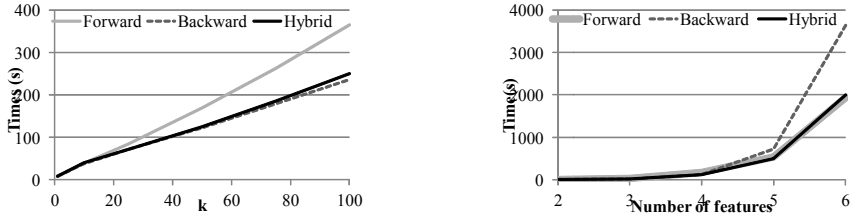
Fig. 10. Comparing the time cost by varying *k* and *d*.

force approach far exceed those of the other three strategies. This demonstrates the importance of a supporting data structure to reduce I/O costs. The time cost of the traditional identification algorithm is far less than that of the brute force approach, due to the inclusion of an index support structure, which greatly reduces access times. Nonetheless, the time cost of the traditional identification algorithm is far greater than that of the three proposed strategies. This can be explained by the fact that the traditional identification algorithm is required to access the entire dataset repeatedly, whereas the three proposed strategies access the entire dataset only once. Further analysis pertaining to the performance of the three strategies is outlined in the following.

### 5.2.2 Comparison of the three proposed strategies

We study in the following the performance of the forward strategy, the backward strategy, and the hybrid strategy. Fig. 11 (a) shows the time cost of the strategies, with the range of *k* varying from 1 to 100. The value of *d* and the number of the patterns are fixed at 4 and 1M, respectively. As we can see, the difference between the forward strategy and the other two strategies increases as *k* grows. This is mainly because before deleting an element *e* the forward strategy has to find *k*+1 patterns better than *e*, but the other two strategies overcome this obstacle. By using Theorems 1 and 4, most elements in the backward strategy can be deleted after the strategy finds a pattern or a few patterns better than them. Hence, both of the number of comparisons and accesses required in the backward strategy are less than those of the forward strategy. This simulation also indicates that the backward strategy and the hybrid strategy are less sensitive to *k* than the forward strategy and hence are suitable for dealing with a request with a greater *k*.

Fig. 11 (b) gives the time cost of the three strategies by varying *d* from two to six. The value of *k* and the number of the patterns in this simulation are fixed at 50 and 1M,




(a) By varying *k*. (b) By varying *d*.

Fig. 11. Comparing the time cost of the three strategies by varying *k* and *d*.

respectively. This figure shows that for all strategies, the time costs increase exponentially with the increase of $d$. This is because the probability that an element is worse than a pattern decreases exponentially as $d$ grows. For example, given a datapool with six features (the values of these features have been normalized to the range [0, 1]) and a pattern $p$(0.3, 0.3, 0.3, 0.3, 0.3, 0.3) in the linked list. When two features are considered, the probability that an element is worse than $p$ is about $(1–0.3)^2 = 0.49$. However, when all six features are considered, this probability becomes $(1–0.3)^6 = 0.12$. This implies that when $d$ increases, finding a pattern better than an element becomes more difficult so that more comparisons between an element and a pattern are required in the searching process. Fig. 11 (b) also reveals that the difference between the backward strategy and the other two strategies increases as the value of $d$ grows. The main reasons are (1) in the backward strategy the comparison of an element with a pattern starts from the one with the highest score; and (2) when $d$ increases, the probability that an element is worse than a pattern with a high score can be quite small. Hence, when $d$ grows, not many patterns better than an element $e$ can be found at the early stage of the backward strategy so that $e$ needs to be compared with the patterns of a lower score, which increases the execution time. As the hybrid strategy always makes the best choice in terms of the comparison order, the time cost of the hybrid strategy is similar to the forward strategy and less than the backward strategy. This simulation demonstrates that the forward strategy and the hybrid strategy are suitable for dealing with a datapool with a greater number of features. According to the above two simulations, we conclude that the hybrid strategy is the best choice among the three strategies, as it incurs the least time cost in all circumstances.

## 6. CONCLUSIONS

In this paper, we proposed a novel mechanism for dealing with the pattern identification problem caused by a vague pattern. This mechanism is able to find suitable answers for a pattern that has different degree of ambiguity in different features. Three strategies, the forward strategy, the backward strategy, and the hybrid strategy, were proposed to enhance the efficiency of the mechanism. A series of theorems was derived so as to accelerate the searching processes. The simulations demonstrated the effectiveness of the proposed mechanism and revealed that the hybrid strategy is the best choice for the proposed mechanism among the three strategies.

## ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Technology of Taiwan, under Contracts MOST 106-2119-M-224-003 and MOST 106-2221-E-006-247.

## REFERENCES

1. T. Ahonen, E. Rahtu, V. Ojansivu, and J. Heikkila, "Recognition of blurred faces using local phase quantization," in *Proceedings of International Conference on Pattern Recognition*, 2008, pp. 1-4.

2. M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski, "Face recognition be independent component analysis," *IEEE Transactions on Neural Network*, Vol. 13, 2002, pp. 1450-1464.
3. R. Chellappa, J. Ni, and V. M. Patel, "Remote identification on faces: problems, prospects, and progress," *Pattern Recognition Letters*, Vol. 33, 2012, pp. 1849-1859.
4. K. Choi, K. A. Toh, and H. Byun, "Incremental face recognition for large-scale social network services," *Pattern Recognition*, Vol. 45, 2012, pp. 2868-2883.
5. B. A. Draper, K. Beak, M. S. Bartlett, and J. R. Beveridge, "Recognition faces with PCA and ICA," *Computer Vision and Image Understanding*, Vol. 91, 2003, pp. 115-137.
6. A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, 2001, pp. 643-660.
7. B. Gunturk, A. Batur, Y. Altunbasak, M. H. I. Hayes, and R. Mersereau, "Eigenface-domain super resolution for face recognition," *IEEE Transactions on Image Processing*, Vol. 12, 2003, pp. 597-606.
8. A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Record*, Vol. 14, 1984, pp. 47-57.
9. Z. He and L. Jin, "Activity recognition from acceleration data based on discrete consine transform and SVM," in *Proceedings of IEEE Conference on Systems*, *Man*, *and Cybernetics*, 2009, pp. 5041-5044.
10. P. Hennings-Yeomans, S. Baker, and B. Kumar, "Simultaneous super-resolution and feature extraction for recognition of low-resolution faces," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1-8.
11. H. Ishida, T. Takahashi, I. Ide, Y. Mekada, and H. Murase, "Recognition of camera-captured low-quality characters using motion blur information," *Pattern Recognition*, Vol. 41, 2008, pp. 2253-2262.
12. A. Nefian and M. Hayes, "An embedded hmm-based approach for face detection and recognition," in *Proceedings of IEEE Conference on Acoustics*, *Speech and Signal Processing*, 1999, pp. 3553-3556.
13. T. H. B. Nguyen and H. Kim, "Novel and efficient pedestrian detection using bidirectional PCA," *Pattern Recognition*, Vol. 46, 2013, pp. 2220-2227.
14. M. Nishiyama, A. Hadid, H. Takeshima, J. Shotton, T. Kozakaya, and O. Yamaguchi, "Facial deblur inference using subspace analysis for recognition of blurred faces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, 2011, pp. 838-845.
15. X. X. Niu and C. Y. Suen, "A novel hybrid CNN-SVM classifier for recognizing handwritten digits," *Pattern Recognition*, Vol. 45, 2012, pp. 1318-1325.
16. N. Pinto, J. J. DiCarlo, and D. D. Cox, "How far can you get with a modern face recognition test set using only simple features?" in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2009, pp. 2591-2598.
17. P. Phillips, "Matching pursuit filters applied to face identification," *IEEE Transactions on Image Processing*, Vol. 7, 1998, pp. 1150-1164.
18. P. Phillips, H. Wechsler, J. Huang, and P. Rauss, "The Feret database and evaluation procedure for face-recognition algorithms," *Image and Vision Computing*, Vol. 16,

1998, pp. 295-306.
19. O. Räsänen and U. K. Laine, "A method for noise-robust context-aware pattern discovery and recognition from categorical sequences," *Pattern Recognition*, Vol. 45, 2012, pp. 606-616.
20. J. Y. Yang, J. S. Wang, and Y. P. Chen, "Using acceleration measurements for activity recognition: an effective learning algorithm for constructing neural classifier," *Pattern Recognition Letters*, Vol. 29, 2008, pp. 2213-2220.
21. Y. Yao, B. Abidi, N. Kalka, N. Schmid, and M. Abidi, "Improving long range and high magnification face recognition: Database acquisition, evaluation, and enhancement," *Computer Vision and Image Understanding*, Vol. 111, 2008, pp. 111-125.
22. M. L. Yiu and N. Mamoulis, "Efficient processing of top-*k* dominating queries on multi-dimensional data," in *Proceedings of International Conference on Very Large Data Bases*, 2007, pp. 483-494.

**Yi-Chung Chen (陳奕中)** received the B.S. and M.S. degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, in 2007 and 2008, and the Ph.D. degree in Department of Computer Science and Information Engineering from National Cheng Kung University, Tainan, Taiwan, in 2014. He joined the faculty of Department of Information Engineering and Computer Science, Feng Chia University in 2014 and participated in some projects related to AI techniques. He is currently an Assistant Professor in the Department of Industrial Engineering and Management, National Yunlin University of Science and Technology. His research interests include spatio-temporal databases, recommendation systems, social network analyses, artificial intelligences, and techniques of Industry 4.0.

**Chiang Lee (李強)** received the BS degree from National Cheng-Kung University, Taiwan, in 1980 and the M.E. and Ph.D. degrees in electrical engineering from the University of Florida, Gainesville, in 1986 and 1989, respectively. He joined IBM Mid-Hudson Laboratories, Kingston, New York, in 1989 and participated in a project working on the design and performance analysis of a parallel and distributed database system. He joined the faculty of National Cheng Kung University in 1990 and is currently a Professor of the Department of Computer Science and Information Engineering. He has published many papers in major journals and conferences.