# An Efficient Algorithm for Point Pattern Matching Using Ordered Lists

HONGBIN ZHANG*, MICHIHIKO MINOH* and KATSUO IKEDA*

Matching two-dimensional point patterns is an important problem in the field of pattern recognition and computer vision. Mathematically, it is a graph or subgraph isomorphism problem, and belongs to the class of NP problems. An efficient algorithm is needed for practical applications. This paper presents an approach for matching point paterns by using ordered lists. The measure of matching error is defined, and a method of searching for pairing points is then discussed. The algorithm uses ordered lists to limit the range searched for pairing points, and avoids exhaustive combination of points. Experimental results show the effectiveness of the proposed algorithm. Several problems that occur in certain applications are analyzed at the end of the paper.

## 1. Introduction

Many kinds of problems in pattern recognition can themselves be represented by two-dimensional (2-D) point patterns. For example, the feature points of a fingerprint image (such as the branch points, and end points) can be described by their coordinates in a reference system. The identification of two fingerprints can be viewed as a problem of matching two 2-D point patterns. According to the statistics of experts, there are up to a hundred such feature points in a fingerprint. However, about ten feature points are enough to match two fingerprints [1, 2].

There are also many kinds of problems that are not inherently point patterns, but that can be equivalently represented by point patterns after appropriate transformations. For example, the boundaries of objects in an image can be approximated by fragments and circular arcs, whose end points constitute a point pattern. Thus, the matching of an object to a model in a computer may be treated as a point pattern matching problem.

As stated above, point pattern matching plays an important role in image understanding. In recent years, a considerable amount of research has been done on this problem [3-9].

The two-dimensional point patterns $U$ and $V$ can be described as follows:

$$U=\{u_1, u_2, \cdots, u_i, \cdots, u_m\} \qquad 1 \leq i \leq m$$
$$V=\{v_1, v_2, \cdots, v_j, \cdots, v_n\} \qquad 1 \leq j \leq n,$$

where $u_i=(x_{ui}, y_{ui})$ and $v_j=(x_{vj}, y_{vj})$ are points of $U$ and $V$, $(x_{ui}, y_{ui})$ and $(x_{vj}, y_{vj})$ are coordinates of $u_i$ and $v_j$ in their coordinate systems, and $m$ and $n$ are the numbers of points in $U$ and $V$, respectively. The matching of $U$ and $V$ is a problem of searching for correspondences between the points of $U$ and those of $V$.

Here, the term *point* may stand for various types of object feature such as the centers of circles or ellipses, the centers of gravity of areas, and the critical points of object boundaries that have high curvature. In practical applications, a *point* usually has an attribute or a type with it, but in order to make the matching algorithm robust, only geometrical information is used in the proposed algorithm. Examples of analysis using the attribute of a point will be given at the end of this paper.

It is well known that the problem of point pattern matching has high computational complexity because of its combinatorial nature. Mathematically, it is a problem of graph or subgraph isomorphism. If no heuristics are used, it becomes an NP problem. Over the past few decades, a lot of research work on point pattern matching has been published: *Simon* et al. used the inter-point distance to match point patterns. Their method is applicable only to patterns that have equal numbers of points [3]. An algorithm using a minimal spanning tree was investigated by *Zahn* [4]. His method is sensitive to the omission and addition of points. *Rosenfeld* et al. proposed a relaxation scheme for point pattern matching [6], and showed that the relaxation approach is more tolerant of global distortion than other methods [5]. However, the convergence speed is slow in their method. *Ogawa* formed a matching method based on a fuzzy relaxation [8], and recently proposed another algorithm using *Delaunay triangulation* [9]. The point pattern is partitioned into a set of triangles,

This is a translation of the paper that appeared originally in Japanese in Transactions of IPSJ, Vol. 31, No. 7 (1990), pp. 1005-1014.

*Department of Information Science, Faculty of Engineering, Kyoto University, Kyoto 606-01, Japan.

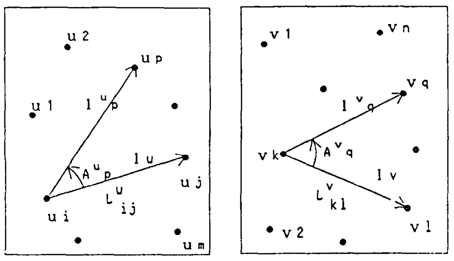

Fig. 1 Definition of Base Line Pair (BLP).

and the largest maximal clique of the consistency graph is used to find the largest set of mutually consistent point pairs. The method is invariant under *affine transformation* of point patterns. But if the *distinguishing points*, which determine the partition, are selected improperly, the matching may fail to work.

In principle, the problem of point pattern matching can be represented by a tree, which we may call a *matching state tree*, and solved by searching the tree. In order to improve the efficiency of point pattern matchings, considerable attention has been paid to the approaches of searching trees and pruning branches [10], but little thought has been given to the method of combining points during construction of the matching state tree. A general method is to pair the points of $U$ and $V$ exhaustively, to examine their consistency, and to determine whether or not to add this pair of points to the tree. Here, we present a much more efficient algorithm for solving the problem of point pattern matching. A distinguishing feature of our algorithm is the use of an ordered list of points to avoid exhaustive combinations of corresponding point pairs. The algorithm can confine the range searched for potential pairings so that the number of pairings needing to be examined is greatly reduced. In Section 2, we introduce some basic definitions and terminology. In Section 3, the problem of pairing points in the matching process is analyzed, and an algorithm based on the ordered list is proposed. In Section 4, some experimental results for synthetic and actual images are shown. In Section 5, the computational complexity of the algorithm is analyzed. In the last section, we make some concluding remarks.

## 2. Definitions

Due to the imperfection of image processing and to noise, the point pattern extracted from an image may contain spurious points and/or may not contain indispensable points. There may also be uncertainty as to the geometrical position of each point. Therefore, the matching of a point pattern extracted from a sensed image to a model is not simply a problem of determining whether the two patterns are exactly the same; rather, it must be performed according to some similarity measure in order to find the matching that has the minimal matching error and the maximal number of corresponding points. We introduce the definition of a Base Line Pair (*BLP*) between $U$ and $V$, and define the matching error based on the *BLP*.

**Definition 1:** [base line $L^u_{ij}$ and $L^v_{kl}$]
Let $u_i$ and $u_j (i<j)$ be points of point pattern $U$. The directed line segment $\overline{u_iu_j}$ is called a base line of $U$ and is denoted by $L^u_{ij}$, and its length by $l_u$ (Fig. 1). The base line $L^v_{kl}$ and length $l_v$ of $V$ are defined in the same way.

**Definition 2:** [base line pair $(L^u_{ij}, L^v_{kl})$]
When $L^u_{ij}$ is matched to $L^v_{kl}$, the pair of $L^u_{ij}$ and $L^v_{kl}$ is called a Base Line pair (*BLP*). The length scale factor $SF$ associated with $(L^u_{ij}, L^v_{kl})$ is defined as $SF=l_u/l_v$.

**Definition 3:** [matching error]
Suppose that $(L^u_{ij}, L^v_{kl})$ is a *BLP* of $U$ and $V$. When a point $u_p$ of $U$ is matched to $v_q$ of $V$, the matching error due to this pairing is defined as follows (Fig. 1): Let $l^u_p$ denote the distance between $u_p$ and $u_i$, and $l^v_q$ the distance between $v_q$ and $v_k$. Let $A^u_p$ denote the angle between $\overline{u_iu_p}$ and $\overline{u_iu_j}$, and $A^v_q$ the angle between $\overline{v_kv_q}$ and $\overline{v_kv_l}$. The matching error of $u_p$ and $v_q$, associated with $(L^u_{ij}, L^v_{kl})$, is defined as

$$E(u_p, v_q; L^u_{ij}, L^v_{kl}) \equiv KA*|A^u_p - A^v_q| + KL*|l^u_p - SF*l^v_q|,$$

where $KA$ and $KL$ are the weights for angle error and length error, respectively.

The reason for defining the scale factor $SF$ is to cope with a change of scale between $U$ and $V$. Using relative distance and relative angle ensures that the algorithm is independent of translation and rotation of point patterns.

## 3. Point Pattern Matching Algorithm

### 3.1 Problems in Searching the Matching State Tree

In principle, the problem of point pattern matching can be described by the matching state tree (Fig. 2). The root node in this representation scheme is the initial state of the matching process. Each leaf node represents a final state that denotes a possible matching of the point patterns. An intermediate node represents an intermediate matching result, which is embedded by a new pair of matched points of $U$ and $V$ into the parent node. The incremental matching error caused by embedding the new pair is associated with the branch connecting these nodes. Thus, the path from the root node to a leaf node that has the minimal sum of the incremental matching errors represents the best point pattern matching.

As stated above, some points of $U$ (or $V$) extracted from an image may be spurious, and/or some indispensable points of $U$ (or $V$) may be missed. Therefore it may not be possible to match some points of $U$ (or $V$) to any points of $V$ (or $U$). In this case, symbols such as $(u_p, \Lambda)$ or $(\Lambda, v_q)$ are employed in the matching process, where $\Lambda$ denotes null. The incremental matching error is given as $KA*A_t+KL*L_e$, whose value is larger

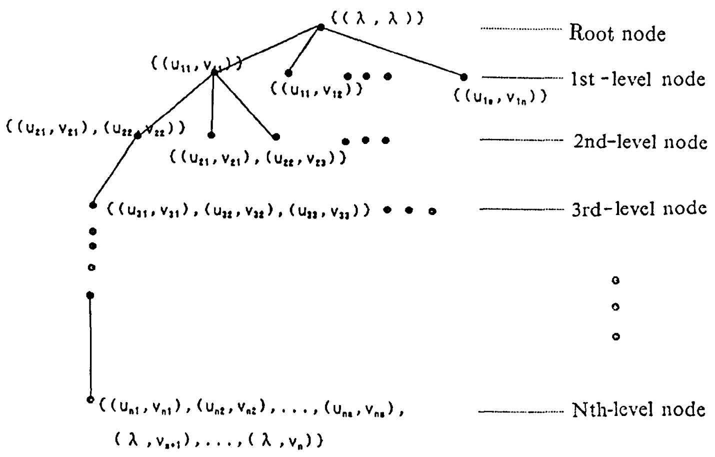

Fig. 2 Matching state tree.

than that of the incremental matching error with a matching point, where $A_t$ is the allowable angle error, and $L_e$ is the distance between $u_p$ (or $v_q$) and the starting point of the base line (for $v_q$, it is multiplied by the scale factor *SF*).

In constructing the matching state tree, it is essential to use geometrical constraints to reduce the size of matching state tree by ruling out incorrect pairing of points as soon as possible. The common approach is to enumerate all of the combinations of points as pairing candidates and then to remove inappropriate pairings by using geometrical constraints. However, little attention has been paid to the method of combining points. We now discuss how the geometrical relations of points can be used to regulate the pairing process and to confine the range searched for pairing candidates in the matching process.

## 3.2 Strategy of the Algorithm

In order to avoid the exhaustive pairing of points of $U$ and $V$, the algorithm is applied to two ordered lists of points of $U$ and $V$. The ordered lists are generated as follows: Let $(L^u_{ij}, L^v_{kl})$ be a base line pair of $U$ and $V$. The points of $U$ (and $V$) other than the starting and ending points of the base line are sorted in ascending order of the size of their angles to the base line (Fig. 3). Associated with *BLP* $(L^u_{ij}, L^v_{kl})$, two ordered lists *UL* and *VL* are thus obtained:

$$UL=\{u_1, u_2, \cdots, u_p, u_{p+1}, \cdots, u_{m-2}\} \quad 1 \le p \le m-2$$
$$VL=\{v_1, v_2, \cdots, v_q, v_{q+1}, \cdots, v_{n-2}\} \quad 1 \le q \le n-2$$

If we suppose that $u_p$ is matched to $v_q$, then the that points that are candidates to match $u_{p+1}$ would be $v_{q+1}$ or some points that follow $v_{q+1}$ and are within the range of the allowable angle error. Let $A_u$ denote the angle of $u_{p+1}$ to $L^u_{ij}$ and $A_t$ the allowable angle error. Only these points of *VL* that satisfy $|A_u - A_v| < A_t$, where $A_v$ is their angle to $L^v_{kl}$, are feasible candidates, and the others can be left out of consideration. Thus, the range searched for corresponding point pairs is quite confined. Exhaustive pairing of points is avoided.

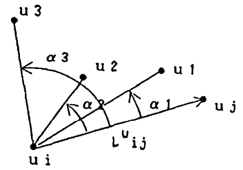

Fig. 3 Sorting points of U by their angles.

In practical applications, for some maximum allowable error $A_t$, the number of points in *VL* that can be matched to one of the points of *UL* can be confined within some constant. Assuming that the order of points is not altered by noise, the matching of *UL* and *VL* can be completed in a time that is linear with respect to the number of elements of *UL*. Even if the order is altered by noise, the size of the matching state tree is considerably decreased. To simplify the analysis, we will assume in the following discussion that the order is not altered by noise.

## 3.3 Algorithm

The flowchart of the algorithm is shown in Fig. 4. A PASCAL-like description of the algorithm is given below.

**Algorithm :**

```
1. Determine all the feasible BLPs between U and V.
2. BEST_ERROR=some maximum value.
   BEST_MAP=null list.
3. FOR(each BLP ($L^u_{ij}$, $L^v_{kl}$))DO;
```

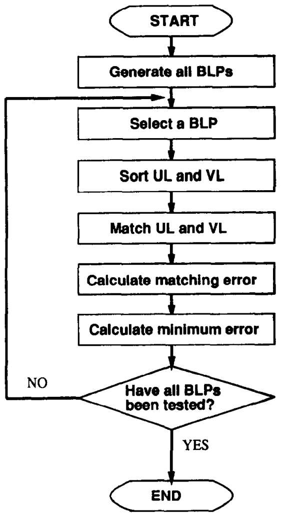

Fig. 4 Flowchart of the algorithm.

(a) For each point of $U$ (and $V$) except the starting and ending points of the base line, calculate the distance to the starting point of $L^u_{ij}$ (and $L^v_{kl}$), and the angle with $\overline{u_i u_j}$ (and $\overline{v_k v_l}$). Sort the points according to the angles and form two ordered lists *UL* and *VL*.

```
(b) $p=1$.
(c) FOR $p$-th element $u_p$ of $UL$, DO;
    i.   Let the angle of $u_p$ with $L^u_{ij}$ be $U\_ANGLE$.
    ii.  Search for the element $v_q$ of $VL$ that has
         not yet been matched and whose angle with
         $L^v_{kl}$, $V\_ANGLE$, is the smallest within the
         allowable angle error.
    iii. WHILE $(ABS(V\_ANGLE - U\_ANGLE) < A_t)$
         IF ($u_p$ and $v_q$ satisfy the matching conditions)
         THEN {
             match $v_q$ to $u_p$;
             calculate the incremental matching error;
             set matched flags to $u_p$ and $v_q$, respectively;
             $p=p+1$;
             IF $(p \leq m-2)$
             THEN GOTO [(c)];
             ELSE GOTO [(d)];
             ENDIF}
         ELSE {
             $v_q$=the element of $VL$ next to $v_q$;
             set the angle of $v_q = V\_ANGLE$}
         ENDIF
         ENDWHILE
    iv.  /*Since there is no point of $VL$ that
         satisfies the matching conditions within the
         allowable angle error, */
         {match $u_p$ to $\Lambda$;
         calculate the incremental matching error;
         $p=p+1$;
         IF $(p \leq m-2)$
         THEN GOTO [(c)]
         ELSE GOTO [(d)]
         ENDIF
         }
(d) Calculate the incremental matching error for the
    elements of $VL$ that have not yet been matched.
(e) Calculate the normalized matching error $N\_ERROR$.
    Hold the matching result in array $MAP$.
    IF $(N\_ERROR < BEST\_ERROR)$
    THEN {
        $BEST\_ERROR = N\_ERROR$;
        $BEST\_MAP = MAP$}
    ENDIF
```

4. If all of the BLPs are tested, end. Otherwise, go to Step 3.

In the above algorithm, *BEST_ERROR* is a variable that records the minimal matching error, and *BEST_MAP* is an array that records the matching result.

The matching conditions for $u_p$ and $v_q$ are as follows:

**CONDITION 1:** $|A_u - A_v| < A_t$

where $A_u$ is the angle of $u_p$ with $L^u_{ij}$, $A_u$ the angle of $v_q$ with $L^v_{kl}$, and $A_t$ the allowable angle error threshold.

**CONDITION 2:** $|L_u - L_v| / L_u < \mathcal{L}_t$

where $L_u$ is the distance between $u_p$ and the starting point of $L^u_{ij}$ and $L_v$ the distance between $v_p$ and the starting point of $L^v_{kl}$. $\mathcal{L}_t$ is the threshold of the allowable relative length error.

As stated in Section 3.1, for those elements of *UL* and *VL* that are not matched, the length error and the angle error are also calculated and added to the total length error *L_ERROR* and the total angle error *A_ERROR*.

The normalized error *N_ERROR* is defined as

$$N\_ERROR = KL * L\_ERROR / LN + KA * A\_ERROR / AN,$$

where *KL* and *KA* are weighting factors for the length error and the angle error, respectively. *LN* and *AN* are normalization constants and are defined as follows:

$$LN = \sum_m (\textit{the length of } u_p) + \sum_n (\textit{the length of } v_q) * SF$$

$$AN = (m+n) * A_t,$$

where $m$ and $n$ are the numbers of points of $U$ and $V$, respectively.

The first step of the algorithm is to generate all the feasible base line pairs. A *BLP* represents an initial partial matching and the algorithm will make the initial matching grow pair by pair. A *BLP* can be created by pairing two points of $U$ with two points of $V$. In practice, one can use compatibility of $L^u_{ij}$ and $L^v_{kl}$ to reduce the number of *BLP*s considerably: if the scale factor of $U$ against $V$ is known in advance, one may use this factor to rule out unfeasible pairings of $L^u_{ij}$ and $L^v_{kl}$. This will greatly reduce the number of *BLP*s.

## 4. Experimental Results

Experiments were done, using synthetic and real data, and showed that our algorithm is very efficient.

**Experiment 1:** Synthetic data obtained by a 2-D random number generator are used. $N$ points are generated to form the point pattern $V$. Then, $m$ points from $V(m<n)$ are picked out randomly and rotated, the scale is changed, and spurious points created by noise are added to form the point pattern $U$. That is, let $v_q$ be a point of $V$, and let its coordinates be $(x, y)$. The coordinates of $u_p$ of $U$, $(x', y')$, the counterpart of $v_q$, are obtained as follows:

$$x' = (x*\cos\beta - y*\sin\beta)*(f+c)$$
$$y' = (x*\sin\beta - y*\cos\beta)*(f+c),$$

where $\beta$ is the angle of rotation around the origin, $f$ is a scale factor, and $c$ represents noise chosen from the normal distribution with mean 0 and standard deviation $\sigma$. Supplying different initial values to the random number generator, we generated seven cases of data with different numbers of points. Each case consists of ten groups of different point patterns. The parameters used in the experiment were as follows:

The allowable relative length threshold $\mathscr{L}_t$ is 0.15.
The allowable angle error $A_t$ is 10.
The standard deviation $\sigma$ is 0.025.

For all of these data, the algorithm successfully found the correct matchings. The average CPU time is shown in Table 1.

For comparison, a general matching tree algorithm was executed on the same data, that is, an algorithm that exhaustively enumerates all of the pairings of points of $U$ and $V$, then uses the distance and angle constraints to exclude unfeasible pairings. The results are shown in Table 2.

A comparison of Table 1 with Table 2 shows that our algorithm is much faster. This is because the search range for pairing points is considerably restricted by the ordered lists.

**Experiment 2:** Various point patterns with different numbers of points are generated by the same method as in Experiment 1 in order to investigate the relation between the number of points and the execution time of the algorithm. As shown in Table 3, the results are satisfactory for problems that consist of a few tens of points.

The relation of the average CPU time to $m*n$, the product of the numbers of points of $U$ and $V$, is shown in Fig. 5. This curve approximately coincides with the time complexity analysis of Case 1 in Section 5.

**Experiment 3:** The algorithm was applied to shape matching of 2-D objects. Figures 6(a) and (b) are polygonal approximations of 2-D shapes. Figures 6(b) shows a model contained in (a). The numbers of points in (a) and (b) are 27 and 16, respectively. The matching result is shown in Fig. 7. In this and later figures, small circles denote correctly matched points, and small triangles points matched with $\Lambda$. The CPU time for matching is 31,128 milliseconds.

Table 1 Average CPU time of the algorithm (ms). (Experiment 1)

| No. of points in $U$ | 5 | 6 | 7 | 6 | 7 | 6 | 8 |
|---|---|---|---|---|---|---|---|
| No. of points in $V$ | 15 | 16 | 15 | 18 | 17 | 21 | 16 |
| Av. time (ms) | 345 | 673 | 850 | 999 | 1,266 | 1,781 | 1,473 |

(on NEC/MS 190)

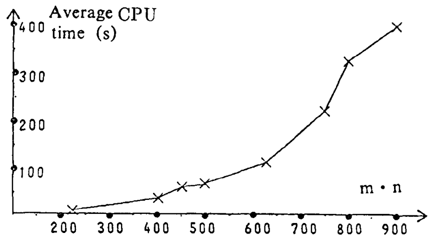

Fig. 5 Curve of CPU time against number of points.

Table 2 Average CPU time of the general matching tree algorithm (ms).

| No. of points in $U$ | 5 | 6 | 7 | 6 | 7 | 6 | 8 |
|---|---|---|---|---|---|---|---|
| No. of points in $V$ | 15 | 16 | 15 | 18 | 17 | 21 | 16 |
| Av. time (ms) | 20,451 | 83,674 | 190,526 | 158,253 | 423,255 | 467,533 | 667,270 |

(on NEC/MS 190)

Table 3 Average CPU time of the algorithm (ms). (Experiment 2)

| No. of points in $U$ | 15 | 20 | 15 | 20 | 25 | 25 | 20 | 30 |
|---|---|---|---|---|---|---|---|---|
| No. points in $V$ | 15 | 20 | 30 | 25 | 25 | 30 | 40 | 30 |
| Av. time (ms) | 7,450 | 39,430 | 59,921 | 72,916 | 118,298 | 224,254 | 325,961 | 399,136 |

(on NEC/MS 190)

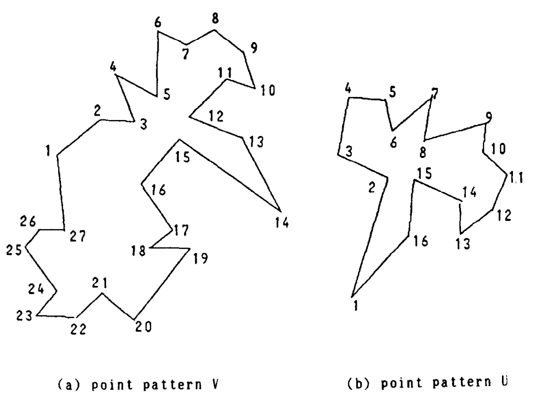

Fig. 6 Matching of 2-D shapes (Experiment 3).

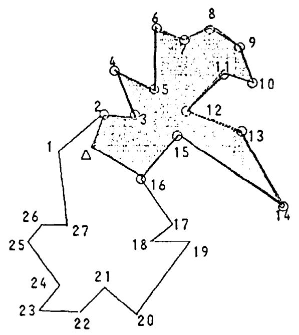

Fig. 7 Matching result of Experiment 3.

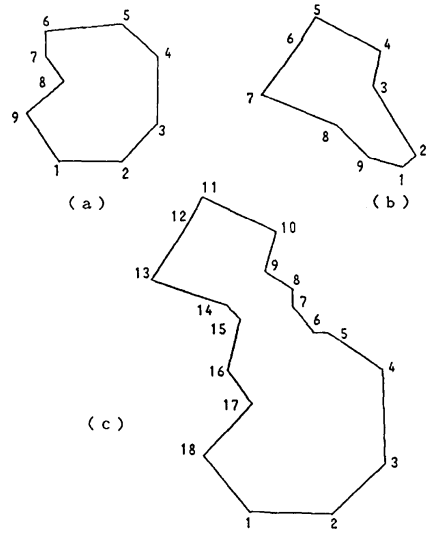

Fig. 8 Matching of 2-D shapes (Experiment 4).

**Experiment 4:** In Fig. 8, two models (a) and (b) occlude each other and form an object shown in Fig. 8(c). Note that the scale of model (a) is changed in the object. The matching result is shown in Fig. 9. The data for this experiment were taken from [11]. Although the experimental environments are different, it seems that our results are much better than those in [11]. Table 4 shows a comparison of our CPU time and experimental environment with those in [11].

**Experiment 5:** Figure 10 shows a synthetic example in which three models (a), (b) and (c) occlude one another and form an object (d). It is necessary to identify each of the models from the object. The algorithm has successfully recognized the models (a),

Table 4 Comparison with Reference 11 (ms).

| | Matching time | | Computer used |
|---|---|---|---|
| | a to c | b to c | |
| Our algor. | 3.67 | 2.40 | NEC/ MS-190, 4.0MIPS |
| Ref. 11 | 52.79 | 42.90 | DEC/ PDP-10, 1.8MIPS |

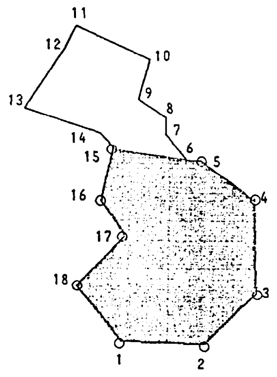

Fig. 9(i) Matching result of Figs. 8(a) and (c).

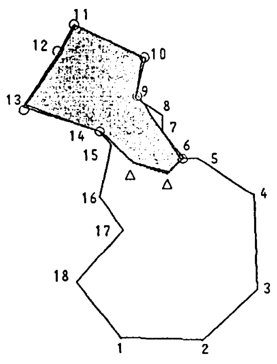

Fig. 9(ii) Matching result of Figs. 8(b) and (c).

(b), and (c) from (d). The result is shown in Fig. 11. The CPU times for matching (a) to (d), (b) to (d), and (c) to (d) are 0.93 s, 0.84 s and 2.40 s, respectively. The total CPU time for matching is 4.17 s. These data are also taken from [11], where the total time for matching was 152.7 s.

**Experiment 6:** Many approaches to recognizing overlapping objects have been proposed in recent years [10, 11]. We applied the point pattern matching algorithm to this kind of problem. Figures 12(a) and (b) represent polygonal approximations of the object boundaries, extracted from the images of two industrial parts, respectively, and (c) is a polygonal approximation of the composite object consisting of (a) and (b). The result of matching (a) and (c) is shown in Fig. 13. The numbers of vertices in (a) and (c) are 21 and 23, respectively. The CPU time for matching is 47.70 s.

In the problem of matching 2-D shapes or recognizing overlapping 2-D objects, there may be another constraint that can be used to accelerate the matching speed further. We investigated the method to utilize the contiguous order and structure of the point patterns, and achieved good results [12].

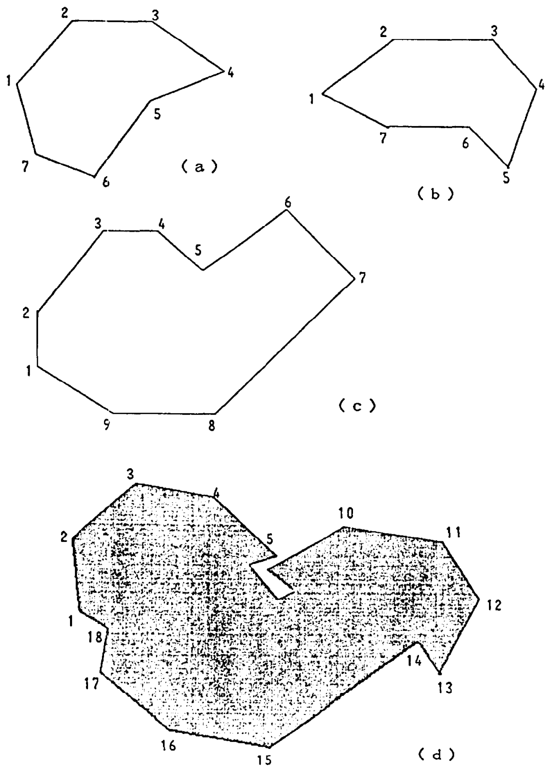

Fig. 10 The problem in Reference 11 (Experiment 5).

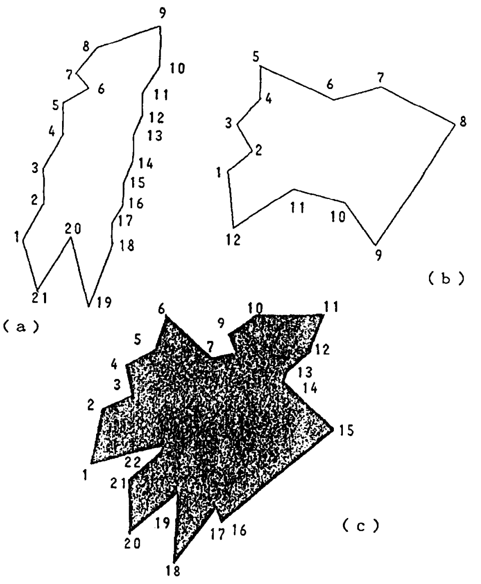

Fig. 12 Recognition of overlapping objects (Experiment 6).

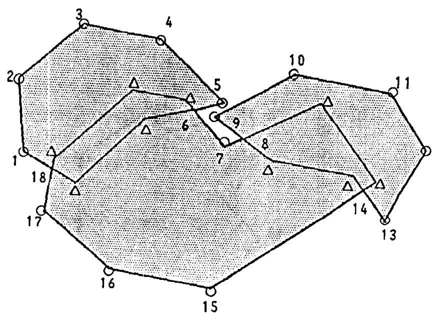

Fig. 11 Results of Experiment 5.

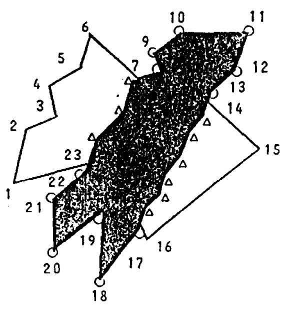

Fig. 13 Result of Experiment 6.

## 5. Computational Complexity Analysis

In this section, we briefly discuss the computational complexity of the proposed algorithm. The time complexity of the algorithm depends on two factors: the number of *BLP*s generated and the cost of sorting and matching operations for each *BLP*. The matching of *UL* and *VL* is bounded by a time linear with respect to the number of elements in *UL*. The sorting of *UL* and *VL* can be completed in $O(m*\log m)$ and $O(n*\log n)$ time, where $m$ and $n$ are the numbers of points of $U$ and $V$, respectively. Therefore, the cost of operations at each *BLP* is bounded by $O(p*\log p)$ time ($p=\max(m, n)$). The number of *BLP*s generated depends on the property of the problem, on the noise level, and on the amount of additional information. If no other information is available, the number of *BLP*s depends mainly on the reliability of points in $U$ and $V$. Here, the reliability means that a point in one point pattern has a corresponding point in the other. Different cases are discussed below.

**Case 1:** If none of the points of $U$ and $V$ is reliable, all of the pairings of base lines of $U$ and $V$ must be tested. In this case, the number of *BLP*s is equal to $m^2n^2$, and the time complexity of the algorithm is $O(m^2n^2*p \log p)$. However, as we will see later, the number of *BLP*s in practical problems is far less than $m^2n^2$, which is the worst case.

**Case 2:** When it is known in advance that a point $u_p$ of $U$ definitely corresponds to a point $v_p$ of $V$, the time complexity is $O(mn*p \log p)$.

**Case 3:** When it is known in advance that a base line $L_{ij}^{u}$ of $U$ is definitely a counterpart of $L_{kl}^{v}$ of $V$, then it is not necessary to test the other *BLP*s. The time complexity is $O(p \log p)$.

**Case 4:** Assume that all of the points of $U$ are reliable, that is, that all of the points definitely have corresponding points in $V$. $V$ definitely has a counterpart for a base line of $U$. The time needed to find the counterparts is bounded by $O(n^2)$. Therefore, the time complexity is $O(n^2 p \log p)$.

It can be seen that the complexity of the algorithm largely depends on the number of *BLP*s to be tested. The proposed algorithm searches for the minimal matching error, BEST_*ERROR*, among all of the potential *BLP*s. However, if the terminating conditions of the algorithm were relaxed, for instance, if the algorithm was terminated when the matching error, *BEST_ERROR*, was less than some threshold, the matching time could be greatly reduced.

## 6. Conclusion

We have proposed an efficient algorithm for 2-D point pattern matching. The measure of matching error has been defined and a method of searching for pairing points discussed. The algorithm employs ordered lists to confine the range searched for pairing, and can thus avoid exhaustive combination of points. The algorithm worked successfully and efficiently for several examples of point patterns. Since the relative distance and angle, as well as the scale factor, are used, the algorithm is invariant with translation, rotation, and scaling. It can be applied to the matching of two general point patterns with different point numbers, and at the same time with spurious and missing points.

We concentrated only on the utilization of the geometrical information contained in point patterns. In practice, however, other information, such as the point attributes and the point types can be incorporated into the algorithm to reduce the number of *BLP*s further and to improve the matching efficiency. The proposed algorithm is simple and could be implemented on parallel processing hardware.

**Acknowledgement**

One of the authors would like to thank Professor Toshiyuki Sakai for his valuable advice and comments on the early stages of this research.

**References**

1. Moayer, B. and Fu, K. S. Syntactic Approach to Fingerprint Pattern Recognition, *Pattern Recognition*, 7 (1975), 1-23.
2. The Science of Fingerprints, FBI Manual, U. S. Govt. Printing Office (1963).
3. Simon, J. C., Csecroun, A. and Roche, C. A Method of Comparing Two Patterns Independent of Possible Transformations and Small Distortions, *Pattern Recognition*, **4**, 1 (1972), 73-81.
4. Zahn, C. T. Graph-Theoretical Methods for Detecting and Describing Gestalt Cluster, *IEEE Trans, Comput.*, **C-20**, 1 (1971), 68-86.
5. Kahl, D. J., Rosenfeld, A. and Danker, A. Some Experiments in Point Matching, *IEEE Trans. Systems*, Man, Cybernet., **SMC-10** (1980), 105-116.
6. Ranade, S. and Rosenfeld, A. Point Pattern Matching by Relaxation, *Pattern Recognition*, **12**, 4 (1980), 269-275.
7. Lavine, D., Lambird, B. A. and Kanal, L. N. Recognition of Spatial Point Patterns, *Pattern Recognition*, **16**, 3 (1983), 289-295.
8. Ogawa, H. Labeled Point Pattern Matching by Fuzzy Relaxation, *Pattern Recognition*, **17**, 5 (1984), 569-573.
9. Ogawa, H. Labeled Point Pattern Matching by Delaunay Triangulation and Maximal Clique, *Pattern Recognition*, **19**, 1 (1986), 35-40.
10. Eric, W., Grimson, L. et al. Localizing Overlapping Parts by Searching the Interpretation Tree, *IEEE Trans. PAMI*, **PAMI-9**, 4 (1987), 469-482.
11. Bhanu, B. and Faugeras, O. D. Shape Matching of Two-Dimensional Objects, *IEEE Trans. PAMI*, **PAMI-6**, 2 (1984), 137-156.
12. Zhang, H., Minoh, M. and Ikeda, K. Recognition of Overlapping Objects Based on Matching the Objects, Boundaries, *Technical Report of IECE of Japan*, PRU88-144 (in Japanese, 1989), 32-40.