A PARALLEL ALGORITHM FOR FINDING CONGRUENT REGIONS*

CHIN-LAUNG LEI and HORNG-TWU LIAW
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106, R.O.C.

Abstract—In this paper, we study the problem for finding all the regions, which are congruent to a testing region $R$, in an input planar figure $F$. In a shared memory system with $m$ processors, we propose an efficient MAX {$O(mn)$, $O(n \log n)$} time parallel algorithm, where $n$, $m$ are the numbers of edges of $F$ and $R$, respectively. Furthermore, our algorithm does not require to read from or write into the same memory location simultaneously, hence it can be implemented on an exclusive-read, exclusive-write (EREW) model.

1. INTRODUCTION
Finding the congruent regions among geometric objects is a popular topic in computational geometry. In general, this problem arises in pattern recognition, computer vision, etc. Recently, some researchers have devoted themselves to investigating this problem [1-4].

Roughly speaking, two planar regions $R$ and $S$ are congruent if there exists a mapping, including a proper geometrical translation and/or rotation, which makes $R$ onto $S$. A formal definition will be given in Section 2. In [3], they defined the congruent regions finding problem as follows: Given a planar figure $F$ and a testing region $R$, determine whether $R$ is congruent to any region of $F$, and then find all of them if they indeed exist. Figure 1 shows an example of this finding problem. Only the shadow regions bounded by edges $(v_0, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_4)$ and bounded by $(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)$ are congruent to the testing region $R$. The value labeled with each edge represents the length of that edge.

The investigation of VLSI technology has made progress in parallel operation that reveals a high degree of parallelism in multiprocessor systems. Basically, there are two different architectural models for multiprocessor systems. One of them is a tightly-coupled system where communication is through a shared memory. Thus, we also say that this system is a shared-memory multiprocessor system. The other one is a loosely-coupled system where communication is done via an interconnection network, that is, this is a message-passing multiprocessor system.

In a shared-memory parallel system, each processor can read from or write into any memory location, depending on whether concurrent read from or concurrent write into a memory is allowed or not. Therefore, a shared-memory parallel system can be further divided into the following four models:

1. Exclusive-Read, Exclusive-Write (EREW) model.

No two processors are allowed to read from or write into the same memory location simultaneously.

2. Concurrent-Read, Exclusive-Write (CREW) model.

Processors are allowed to read from the same memory location, but no two processors are allowed to write into the same memory location simultaneously.

3. Exclusive-Read, Concurrent-Write (ERCW) model.

Processors are allowed to write into the same memory location but no two processors are allowed to read from the same memory location simultaneously.

4. Concurrent-Read, Concurrent-Write (CRCW) model.

Multiple processors are allowed to read from and write into the same memory location simultaneously.

Among the schemes [1-4] mentioned above, only the method proposed in [3] adopts a parallel approach to solve this problem. In this paper, we propose a method rather in EREW model than in CREW model to find the congruent regions. Our algorithm requires only MAX {$O(mn)$, $O(n \log n)$} computation time, where $n, m$ are the numbers of edges of the input planar figure $F$ and the testing region $R$, respectively. Furthermore, Shih, Lee, and Yang's [3] results may contain some repetitive congruent regions. In our algorithm, we have solved this problem. The rest of this paper is organized as follows: In Section 2, some essential definitions and notations of the geometric objects are described. In Section 3, we propose an efficient parallel algorithm for finding the congruent regions and analyze the time complexity. Finally, concluding remarks are given in Section 4.

2. PRELIMINARIES
Before we embark on our study of an efficient parallel algorithm for finding congruent regions based on a shared-memory system, we first give some definitions, notations, and properties of the geometric objects.

A graph $G = (V, E)$ consists of a set $V$ of elements called vertices and a set $E$ of unordered pairs of members of $V$ called edges. The vertices of the graph are shown as points, while the edges are shown as lines connecting pairs of points. The edge between the pair of vertices $v_a$ and $v_b$ is denoted by $(v_a, v_b)$. Here, we call the vertices $v_a$ and $v_b$ the endpoints of the edge $(v_a, v_b)$, and we say the edge $(v_a, v_b)$ is incident with the vertices $v_a$ and $v_b$. In addition, if an edge does not
exhibit a direction, it is called an undirected edge. Thus, an undirected edge $e = (v_a, v_b)$ can also be represented by $(v_b, v_a)$, that is, $(v_a, v_b) = (v_b, v_a)$. On the other hand, a directed edge has only one direction. Let $\bar{e} = \langle v_a, v_b \rangle$ be a directed edge, where $v_a$ is the tail vertex and $v_b$ is the head vertex. Therefore, $\langle v_a, v_b \rangle \neq \langle v_b, v_a \rangle$. A path of directed edges is a sequence of directed edges that are connected with their endpoints. Without loss of generality, let $\bar{P} = [\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_m]$ denote a path of directed edges, where $\bar{e}_i = \langle v_{i-1}, v_i \rangle$, and $v_i \neq v_{i-1}$ when $i \neq j$. Besides, a region consists of a directed cycle and the interior of directed cycle, which is the right-hand side when we walk along the directed edge clockwise. Here, we renewly define the angle of each directed edge. For simplicity, let $\bar{e} = \langle v_1, v_2 \rangle$. First, we translate this directed edge and let the tail vertex $v_1$ be at the Cartesian coordinate origin, that is, the tail vertex $v_1$ is regarded as a Cartesian coordinate origin. The angle between $x$-axis and $\bar{e}$ (or simply the angle of $\bar{e}$) is defined as the angle lying to the right of us when we walk along the positive $x$-axis to the tail of $\bar{e}$. In this paper, we use $A(\bar{e})$ to denote it. Figure 2 shows the edge $\bar{e} = \langle v_1, v_2 \rangle$ and the angle $A(\bar{e})$. In addition, the angle between two adjacent edges $\bar{e}_1 = \langle v_1, v_2 \rangle$ and $\bar{e}_2 = \langle v_2, v_3 \rangle$, denoted by $A(\bar{e}_1, \bar{e}_2)$, is defined as the angle lying to the right of us when we walk along $\bar{e}_1$ to the tail of $\bar{e}_2$. Furthermore, let $\bar{e}_1 = \langle v_1, v_2 \rangle$, and $\bar{e}_2 = \langle v_2, v_3 \rangle$, be the opposite direction of $\bar{e}_1$ and $\bar{e}_2$, respectively. If there exists a condition such that $|\bar{e}_1| = |\bar{e}_2|$, $|\bar{e}_2| = |\bar{e}_3|$ and $A(\bar{e}_1, \bar{e}_2) = A(\bar{e}_2, \bar{e}_3)$, then we say that $\bar{e}_1$ is connected forwardly to $\bar{e}_2$ with respect to $\bar{e}_3$, and similarly, $\bar{e}_2$ is connected backwardly to $\bar{e}_1$. Additionally, if two regions $A = [\bar{e}_{A0}, \bar{e}_{A1}, \ldots, \bar{e}_{Am}]$ and $B = [\bar{e}_{B0}, \bar{e}_{B1}, \ldots, \bar{e}_{Bm}]$ are congruent, then there must exist some $i$ such that $\bar{e}_{A_i} = \bar{e}_{B_i}$. In this paper, our approach is inspired by the method found in [3] but improves upon it. In the next section, we present an efficient parallel algorithm for finding all the congruent regions. In [3], they have shown that a path of directed edges $[\bar{e}_0, \bar{e}_1, \ldots, \bar{e}_{m-1}]$ with $|\bar{e}_i| = |\bar{e}_j|$, $0 \leq i \leq m - 1$, and $A(\bar{e}_i, \bar{e}_{i+1}) = A(\bar{e}_j, \bar{e}_{j+1})$, ...

Fig. 1. The input figure $F$ and the testing region $R$.

Fig. 2(a). The directed edge $\bar{e} = \langle v_1, v_2 \rangle$; (b) The angle $A(\bar{e})$ of the directed edge $\bar{e}$.

Fig. 2(a). The directed edge $\bar{e}$, and the testing region $R$. Fig. 3(a). The adjacent edges $\bar{e}_1$ and $\bar{e}_2$; (b) the angle $A(\bar{e}_1)$ of the directed edge $\bar{e}_1$; (c) The angle $A(\bar{e}_2)$ of the directed edge $\bar{e}_2$; (d) the angle $A(\bar{e}_1, \bar{e}_2)$ between the adjacent edges $\bar{e}_1$ and $\bar{e}_2$. Fig. 3(a). The adjacent edges $\bar{e}_1$ and $\bar{e}_2$; (b) the angle $A(\bar{e}_1)$ of the directed edge $\bar{e}_1$; (c) The angle $A(\bar{e}_2)$ of the directed edge $\bar{e}_2$; (d) the angle $A(\bar{e}_1, \bar{e}_2)$ between the adjacent edges $\bar{e}_1$ and $\bar{e}_2$. Proof. Since $\bar{e}_1$ and $\bar{e}_2$ intersect at the vertex $v_2$, the angle between them is equal to the angle between $\bar{e}_1$ and $\bar{e}_2$, that is, $A(\bar{e}_1, \bar{e}_2) = A(\bar{e}_1) - A(\bar{e}_2)$. From our assumption, any angle must be nonnegative and less than $360^\circ$. Hence, $A(\bar{e}_1, \bar{e}_2) = (A(\bar{e}_2) - A(\bar{e}_1)) \mod 360^\circ$. 

Theorem 2. If $A(\bar{e}_1, \bar{e}_2)$ and $A(\bar{e}_1)$ are given, then $A(\bar{e}_2) = (A(\bar{e}_1) + A(\bar{e}_2)) \mod 360^\circ$.

Proof. From theorem 1, $A(\bar{e}_1, \bar{e}_2) = (A(\bar{e}_2) - A(\bar{e}_1)) \mod 360^\circ$. Thus, $A(\bar{e}_2) - A(\bar{e}_1) = A(\bar{e}_1) + 360^\circ k$, for some integer $k$. That is, $A(\bar{e}_2) = A(\bar{e}_1) + A(\bar{e}_2) + 360^\circ$. Since $A(\bar{e}_2)$ is also nonnegative and less than $360^\circ$, $A(\bar{e}_2) = (A(\bar{e}_1) + A(\bar{e}_2) + 360^\circ) \mod 360^\circ$. 

Figure 3 shows the adjacent edges $\bar{e}_1 = \langle v_1, v_2 \rangle$, $\bar{e}_2 = \langle v_2, v_3 \rangle$, and the angles of $A(\bar{e}_1, \bar{e}_2)$, $A(\bar{e}_1, \bar{e}_2)$. Furthermore, we use $|\bar{e}|$ to represent the length of any directed edge $\bar{e}$ in this paper.
A parallel algorithm for finding congruent regions

3. AN EFFICIENT PARALLEL ALGORITHM

We are now ready to propose a parallel algorithm on an EREW shared-memory computer. The algorithm presented as algorithm Parallel-Congruent makes the following assumptions. Throughout this paper, we use \( m \) processing elements (PEs) to execute our algorithm, where \( m \) is the number of edges of testing region \( R \). Initially, we duplicate any edge of the input planar figure \( F = \{ e_0, e_1, \ldots, e_{m-1} \} \) into two directed edges with opposite direction. Thus, we get a new set \( D = \{ e_0, e_1, \ldots, e_{2m-1} \} \). In addition, the edge \( \tilde{t}_i \) of the testing region \( R = \{ t_0, t_1, \ldots, t_{m-1} \} \) is stored into \( PE_i \), where \( i = 0, 1, \ldots, m-1 \). After some steps of our algorithm, there are \( m \) sets of directed edges \( D_0, D_1, \ldots, D_{m-1} \) stored in shared memory, where \( D_i = \{ e_j : |t_j| = |t_{i,j}|, 0 \leq j \leq m-1 \} \). Each element of \( D_i \) is denoted as \( D_i[k] \), \( 1 \leq k \leq n_i \), where \( n_i \) is the number of edges in \( D_i \) and each \( D_i[k] \) contains six fields EDGE, TAIL, HEAD, ANGLE, FLINK, and BLINK. The fields TAIL and HEAD store the tail vertex and head vertex of the edge, respectively. The field ANGLE stores the angle of the edge in \( D_i \). The field FLINK stores the relative angle of each edge in \( D_i \) to the edge in \( D_{i+1} \). The field BLINK stores the angle of the edge in \( D_i \) to the edge in \( D_{i-1} \). The algorithm proceeds in stages.

Algorithm Parallel-Congruent

Input: The set of edges \( E = \{ e_0, \ldots, e_{m-1} \} \) of the input planar figure \( F \) and the cycle \( \tilde{t}_0, \tilde{t}_1, \ldots, \tilde{t}_{m-1} \) of the testing region \( R \).

Output: All the regions of planar figure \( F \) that are congruent to the testing region \( R \).

Step 1: Duplicate the set of edges \( E = \{ e_0, e_1, \ldots, e_{m-1} \} \) of \( F \) to the set of directed edges \( D = \{ \tilde{e}_0, \tilde{e}_1, \ldots, \tilde{e}_{2m-1} \} \).

Step 2: Load \( \tilde{t}_j \) into the processing element \( PE_i \), where \( j = 0, 1, \ldots, m-1 \).

Step 3: Calculate the values \( FA_i = A(\tilde{t}_i, \tilde{t}_{(i+1) \mod m}) \) and store it into \( PE_i \), where \( i = 0, 1, \ldots, m-1 \).

Step 4: Load the set of directed edges \( D = \{ \tilde{e}_0, \tilde{e}_1, \ldots, \tilde{e}_{2m-1} \} \) into the processing elements by the following operations.

For \( i = 0 \) to \( 2n-1 \) do

For \( j = 0 \) to \( m-1 \) do in parallel

\[ k_j = (j+i) \mod 2n \]

\( \tilde{e}_k \) enters \( PE_i \)

If \( |\tilde{e}_k| = |\tilde{t}_i| \) then \( PE_i \) copies \( \tilde{e}_k \) into the set \( D_i \)

end

Step 5: For \( j = 0 \) to \( m-1 \) do in parallel

Calculate the relative angle of each edge in \( D_i \) and let the value of the angle be \( A_j(\tilde{e}_k), 0 \leq k \leq 2n-1 \).

end

Step 6: For \( j = 0 \) to \( m-1 \) do in parallel

Sort the pairs \( (v_0, A_0(\tilde{e}_k)), 0 \leq k \leq 2n-1 \), where \( v_0 \) is the tail vertex of edge \( \tilde{e}_k \) in \( D_j \).

end

Step 7: For \( j = 0, 2, 4, \ldots, m-2 \) do in parallel

(7.1) For \( k = 1 \) to \( n_j \) do in parallel

\[ A_j(\tilde{e}_k) = (A_j(\tilde{e}_k) - 180^\circ) \mod 360^\circ \]

\[ A_j+1(\tilde{e}_k) = (A_j + A_j(\tilde{e}_k)) \mod 360^\circ \]

// \( 0 \leq k \leq 2n-1 \)

end

end

end

end

(9.1) For \( j = 0, 2, 4, \ldots, m-2 \) do in parallel

(a) and (b) in parallel

(a) For \( k = 1 \) to \( n_j \) do in parallel

If \( D_{j+1}[D_i[k].FLINK].EDGE = 0 \)

then \( D_i[k].EDGE \) \( \leftarrow 0 \)

end

(b) For \( k = 1 \) to \( n_{j+1} \) do

If \( D_{j+1}[D_i[k].BLINK].EDGE = 0 \)

then \( D_{j+1}[2k].EDGE \) \( \leftarrow 0 \)

end

end

end

end

end

(9.2) For \( j = 1, 3, 5, \ldots, m-3 \) do in parallel

Perform step (7.1) of Step 7.

Step 9: All processing elements perform this step iteratively until there is no more edge in each \( D_j \), \( 0 \leq j \leq m-1 \), which can be deleted in one iteration.

(9.1) For \( j = 0, 2, 4, \ldots, m-2 \) do in parallel

(a) For \( k = 1 \) to \( n_j \) do

If \( D_{j+1}[D_i[k].FLINK].EDGE = 0 \)

then \( D_i[k].EDGE \) \( \leftarrow 0 \)

end

(b) For \( k = 1 \) to \( n_{j+1} \) do

If \( D_{j+1}[D_i[k].BLINK].EDGE = 0 \)

then \( D_{j+1}[2k].EDGE \) \( \leftarrow 0 \)

end

end

end

end

end
quires O(m) computation time. Similarly, Step 3 re-
duced edges of F. In Step 2, the directed edges \( t \) to, \( t_1 \) ..... \( t_{m-1} \) are loaded into the processing elements that re-
quires O(m) computation time. Similarly, Step 3 re-
quires O(m) time to calculate the relative angles of all the
directed edges \( \bar{t}_0, \bar{t}_1, \ldots, \bar{t}_{m-1} \). In Step 4, the
set of directed edges \( D = \{ \bar{e}_0, \bar{e}_1, \ldots, \bar{e}_{m-1} \} \) are loaded
into the processing elements and compared with the
directed edge \( \bar{t}_j \) on \( PE_j \), where \( j = 0, 1, \ldots, m-1 \).
Since there exist \( 2n \) directed edges, this step requires
O(n) computation time. In the worst case, there exist
O(n) directed edges in \( D_j \) when Step 4 is completed.
In Step 5, each processing element \( PE_j, j = 0, 1, \ldots,
m-1 \), calculates the relative angles of all the edges in
\( D_j \). Since there may exist O(n) edges in \( D_j \) in the worst
case, it takes O(n) computation time. Step 6 sorts all
the pairs in \( D_j \), where each pair is composed by a tail
vertex and an angle of certain edge. From \([5]\) we know
that any algorithm for sorting \( n \) elements must require
at least \( O(n \log n) \) operations in the worst case. In this
step, we sort the tail vertices in first pass, and then sort
the angles in second pass. Each of these two passes is
completed in \( O(n \log n) \) time. Therefore, the whole
step takes \( O(n \log n) \) time. In Step 7, for each edge in
\( D_j, j = 0, 2, \ldots, m-2 \), it uses binary search scheme
to find the ordered pair \( (v, \theta_{j+1}(\bar{e}_{j+1})) \) in \( D_{j+1} \), where \( v \) is not only the head vertex in \( D_j \) but also the tail
vertex in \( D_{j+1} \), and \( \theta_{j+1}(\bar{e}_{j+1}) \) is the angle of certain
edge in \( D_{j+1}, 0 \leq \theta_{j+1} \leq 2n-1 \). First, we adopt binary
search to find \( \theta_{j+1}(\bar{e}_{j+1}) \) in \( D_{j+1} \) again. Since there may exist
O(n) edges in \( D_{j+1} \) in the worst case, it requires \( O(n \log n) \) time to search the tail vertex and the angle of certain
edge in \( D_{j+1} \). Therefore, the whole step requires \( O(n \log n) \) computation time. Step 8 is similar to Step 7,
and it takes \( O(n \log n) \) computation time. In Step 9,
we use the concept of the odd-even transposition
sort\([6, 7]\) to delete some unsuitable edges in \( D_j \). Shih,
Lee, and Yang [3] have shown that this step must per-
form \( O(m) \) iterations in the worst case to complete
this odd-even scheme. Therefore, this step takes \( O(mn) \)
computation time in the worst case. Furthermore, there
may exist O(n) remainder regions after Step 9. Thus,
using \( m \) processing elements, Step 10.1 requires \( O(n) \)
computation time to feed these regions into \( A_0, \ldots, A_k \),
\( 0 \leq k \leq 2n-1 \), in shared memory. Step 10.2
sorts all the remainder regions after Step 9. Thus,
\( A_0, \ldots, A_k \) with the first vertex.
(10.4) Erase the repetitive regions and
output all the proper congruent regions.

Now, let us discuss this algorithm. The algorithm
Parallel-Congruent uses \( m \) processing elements, where
\( m \) is the number of edges of the testing region \( R \). Step 1
requires \( O(n) \) time to duplicate the set of edges of
the input planar figure \( F \), where \( n \) is the number of
edges of \( F \). In Step 2, the directed edges \( [t_0, t_1, \ldots, t_{m-1}] \) are loaded
into the processing elements that re-
quires \( O(m) \) computation time. Similarly, Step 3 re-
quires \( O(m) \) time to calculate the relative angles of all the
directed edges \( [\bar{t}_0, \bar{t}_1, \ldots, \bar{t}_{m-1}] \). In Step 4, the
set of directed edges \( D = \{ \bar{e}_0, \bar{e}_1, \ldots, \bar{e}_{m-1} \} \) are loaded
into the processing elements and compared with the
directed edge \( \bar{t}_j \) on \( PE_j \), where \( j = 0, 1, \ldots, m-1 \).
Since there exist \( 2n \) directed edges, this step requires
\( O(n) \) computation time. In the worst case, there exist
\( O(n) \) directed edges in \( D_j \) when Step 4 is completed.
In Step 5, each processing element \( PE_j, j = 0, 1, \ldots,
m-1 \), calculates the relative angles of all the edges in
\( D_j \). Since there may exist \( O(n) \) edges in \( D_j \) in the worst
case, it takes \( O(n) \) computation time. Step 6 sorts all
the pairs in \( D_j \), where each pair is composed by a tail
vertex and an angle of certain edge. From \([5]\) we know
that any algorithm for sorting \( n \) elements must require
at least \( \Omega(n \log n) \) operations in the worst case. In this
step, we sort the tail vertices in first pass, and then sort
the angles in second pass. Each of these two passes is
completed in \( O(n \log n) \) time. Therefore, the whole
step takes \( O(n \log n) \) time. In Step 7, for each edge in
\( D_j, j = 0, 2, \ldots, m-2 \), it uses binary search scheme
to find the ordered pair \( (v, \theta_{j+1}(\bar{e}_{j+1})) \) in \( D_{j+1} \), where \( v \) is not only the head vertex in \( D_j \) but also the tail
vertex in \( D_{j+1} \), and \( \theta_{j+1}(\bar{e}_{j+1}) \) is the angle of certain
edge in \( D_{j+1}, 0 \leq \theta_{j+1} \leq 2n-1 \). First, we adopt binary
search to find \( \theta_{j+1}(\bar{e}_{j+1}) \) in \( D_{j+1} \) again. Since there may exist
\( O(n) \) edges in \( D_{j+1} \) in the worst case, it requires \( O(\log n) \) time to search the tail vertex and the angle of certain
edge in \( D_{j+1} \). Therefore, the whole step requires \( O(\log n) \) computation time. Step 8 is similar to Step 7,
and it takes \( O(\log n) \) computation time. In Step 9,
we use the concept of the odd-even transposition
sort\([6, 7]\) to delete some unsuitable edges in \( D_j \). Shih,
Lee, and Yang [3] have shown that this step must per-
form \( O(m) \) iterations in the worst case to complete
this odd-even scheme. Therefore, this step takes \( O(mn) \)
computation time in the worst case. Furthermore, there
may exist \( O(n) \) remainder regions after Step 9. Thus,
using \( m \) processing elements, Step 10.1 requires \( O(n) \)
computation time to feed these regions into \( A_0, \ldots, A_k \),
\( 0 \leq k \leq 2n-1 \), in shared memory. Step 10.2
sorts all the remainder regions after Step 9. Thus,
\( A_0, \ldots, A_k \) with the first vertex. Many papers have been published on par-
allel sorting scheme for last three decades. Whichever
we choose, this step does not exceed \( O(n \log n) \) com-
putation time. Moreover, there may still exist \( O(n) \)
regions after Step 10.3. Thus, it takes \( O(mn) \) com-
putation time to erase and sequentially output all the
congruent regions. Consequently, \( \max\{O(mn), O(n \log n)\} \) time is required in Step 10.
From the above analysis, we observe that our al-
gorithm is dominated by Step 6, 7, 8, 9, and 10. Thus,
the time complexity of this proposed algorithm is
\( \max\{O(mn), O(n \log n)\} \).

The following example illustrates how the congruent
regions are found by the algorithm Parallel-Congruent.

![Fig. 4(a). The input planar figure F; (b) The testing region R.](image1)

![Fig. 5. All the directed edges of figure F.](image2)

![Fig. 6. The contents of D0, D1, and D2.](image3)
A parallel algorithm for finding congruent regions

Example

Figure 4(a) shows an input planar figure F, and Fig. 4(b) shows a testing region R. By executing the Algorithm Parallel-Congruent, we easily find all the regions that are congruent to the testing region R. The input planar figure F consists of a set of edges \( E = \{ e_0, e_1, e_2, e_3, e_4, e_5 \} \) and the testing regions R is defined as \([t_0, t_1, t_2]\). For simplicity, assume that \(|e_0| = |e_1|
= |e_2| = |e_3| = |e_4| = |e_5| = |t_0| = |t_1| = |t_2| = 1.

Step 1 duplicates the set of edges \( E = \{ e_0, e_1, \ldots, e_5 \} \) of the input figure F to the set of directed edges \( D = \{ \vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11} \} \). Figure 5 shows all the directed edges of figure F.

In Step 2, all the directed edges \( t_0, t_1, \) and \( t_2 \) are loaded into the processing elements \( PE_0, PE_1, \) and \( PE_2, \) respectively.

Step 3 calculates the angle of \( FA_0 = A(t_0, t_1) = A(t_1) - (A(t_0) - 180° + 360°), FA_1 = A(t_1, t_2) = A(t_2) - (A(t_1) - 180°), \) and \( FA_2 = A(t_2, t_0) = A(t_0) - (A(t_2) - 180°) + 360°. \) Without loss of generality, let \( FA_0 = 240° - (0 - 180° + 360°) = 60°, FA_1 = 120° - (240° - 180°) = 60°, \) and \( FA_2 = 0 - (120° - 180°) + 360° = 360°. \) Then, we save \( FA_0, FA_1, \) and \( FA_2 \) into \( PE_0, PE_1, \) and \( PE_2, \) respectively. In Step 4, if \|e_k\| = \|t_j\| then \( PE_j \) copies \( \vec{e}_k \) into the set \( D_j, \) where \( 0 \leq k \leq 11, 0 \leq j \leq 2. \) Therefore, \( D_0 = \{ \vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11} \}, D_1 = \{ \vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11} \}, \) and \( D_2 = \{ \vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11} \}. \)

Fig. 7. The result of Fig. 6 after sorting.

Fig. 8. Tables of \( D_0, D_1, \) and \( D_2 \) that contain FLINK and BLINK.
\( \bar{e}_1 \) \) and \( D_2 = \{ \bar{e}_0, \bar{e}_1, \ldots, \bar{e}_{11} \} \). Step 5 calculates the relative angle of each edge in \( D_j \), \( 0 \leq j \leq 2 \), and let the angle be \( A_j(\bar{e}_k) \), \( 0 \leq k \leq 11 \). For simplicity, we show them in terms of three tables, each table contains four fields \( \text{EDGE}, \text{TAIL}, \text{HEAD}, \) and \( \text{ANGLE} \). The field \( \text{EDGE} \) represents the edge name in \( D_j \), \( 0 \leq j \leq 2 \). The fields \( \text{TAIL} \) and \( \text{HEAD} \) are the tail vertex and head vertex of certain edge, respectively. Furthermore, the field \( \text{ANGLE} \) stores the relative angle. In order to continue our example, each angle in field \( \text{ANGLE} \) is given a suitable value. Figure 6 shows these three tables.

In Step 6, we sort all the pairs \( (v_{k,j}, A_j(\bar{e}_k)) \), \( 0 \leq k \leq 11 \), where \( v_{k,j} \) is the tail vertex of edge \( \bar{e}_k \) in \( D_j \), \( 0 \leq j \leq 2 \). Figure 7 shows the result, after sorting, of Fig. 6.

In Steps 7 and 8, binary search scheme is adopted to find the appropriate pair with respect to the testing region \( R \). Two pointers \( \text{FLINK} \) and \( \text{BLINK} \) are used to point to the suitable edge, respectively. For ease and simplicity, we replace these pointers with appropriate edges. Figure 8 shows this result. Furthermore, in Step 9, the scheme of the odd-even transposition sort is adopted to delete unsuitable edges in each \( D_j \) at each iteration until no more edge can be deleted in each \( D_j \).

In Step 10, all the remainder regions are loaded into three sets \( A_0, A_1 \), and \( A_2 \), that is, \( A_0 = \{ \bar{e}_0, \bar{e}_1, \bar{e}_2 \} \), \( A_1 = \{ \bar{e}_1, \bar{e}_2, \bar{e}_0 \} \), and \( A_2 = \{ \bar{e}_2, \bar{e}_0, \bar{e}_1 \} \). Then, these three sets \( A_0, A_1 \), and \( A_2 \) are rotated, respectively. Therefore, \( A_0 = \{ \bar{e}_0, \bar{e}_1, \bar{e}_2 \} \), \( A_1 = \{ \bar{e}_0, \bar{e}_1, \bar{e}_2 \} \), and \( A_2 = \{ \bar{e}_0, \bar{e}_1, \bar{e}_2 \} \). Now, these three sets are sorted in terms of the first vertex. Finally, we erase the possible repetitive sets and output all the remainder regions, which are congruent to the testing region \( R \). Therefore, only the region bounded by edges \( \bar{e}_0, \bar{e}_1, \) and \( \bar{e}_2 \) is congruent to the testing region \( R \).

4. CONCLUDING REMARKS

In this paper, we propose an efficient parallel algorithm for finding the congruent regions. Compare with previous papers. In [8, 9], they limit the input form to be polygons. In [3], it adopts a CREW shared-memory model. In Step 4 of our algorithm, each processing element \( PE_j \), \( j = 0, 1, \ldots, m - 1 \), read the different edge \( \bar{e}_i \), \( i = 0, 1, \ldots, 2n - 1 \), on the same time. Therefore, we modify their CREW model to EREW model in this paper. Furthermore, in [3], their proposed algorithm requires \( O(n^2) \) computation time for finding the congruent regions by using \( m \) processing elements. Whereas, the time complexity of our algorithm only takes \( \text{MAX} \{ O(mn), O(n \log n) \} \) computation time using the same number of processing elements. In addition, the results in [3] may contain some repetitive congruent regions. Here, we have solved this serious problem completely in our proposed algorithm. Moreover, our algorithm can be easily modified to handle the situation where the number of available processing elements is not the same as the number of edges in the testing region. Up until now, there still existed some interesting open problems, for instance, can we propose a parallel algorithm for finding all the 3-dimensional regions, which are congruent to a planar or 3-dimensional testing regions \( R \)? In addition, there is a region similarity problem, which is similar to this congruent region problem, in computational geometry. The difference between them is that the similarity problem must also consider the scaling factors [1, 8, 9, 10].

Acknowledgement—The authors would like to thank the referees for their valuable comments and suggestions.

REFERENCES