Optimal Circular Arc Representations

Lin Chen
FRL, P. O. Box 18345, Los Angeles, CA 90018

Abstract. We investigate some properties of minimal interval and circular arc representations and give several optimal parallel recognition and construction algorithms. We show that, among other things, given an $s \times t$ interval or circular arc representation matrix,

– deciding if the representation is minimal can be done in $O(\log s)$ time with $O(st/\log s)$ EREW PRAM processors, or in $O(1)$ time with $O(st)$ Common CRCW PRAM processors;
– constructing a minimum interval representation can be done in $O(\log(st))$ time with $O(st/\log(st))$ EREW PRAM processors, or in $O(\log t/\log\log t)$ time with $O(st\log\log t/\log t)$ Common CRCW PRAM processors, or in $O(1)$ time with $O(st)$ BSR processors.

1 Preliminaries

Circular arc graphs are well-known intersection graphs and properly contain interval graphs. Benzer [4] showed overlap data involving fragments of a certain gene could be modeled by intervals. This finding confirmed the hypothesis that DNA has a linear structure within genes and helped him win a Nobel Prize. Circular arc graphs also find applications in some other areas such as register allocation. The best way to allocate registers corresponds to an optimal coloring of an interference graph which is often a circular arc graph or even an interval graph (see, *e.g.*, [16]). Many algorithms on circular arc graphs work on circular arc representations (see, *e.g.*, [15]), which can be constructed from circular arc graphs (see, *e.g.*, [9]). In this paper, we study the properties of minimal interval and circular arc representations and present some efficient recognition and construction algorithms.

Given a family S of nonempty sets, the intersection graph G has vertices corresponding to the sets in S and two distinct vertices of G are adjacent iff the corresponding sets in S intersect. S is called an *intersection representation* (IR) for G. If S is a family of arcs on a circle, G is called a *circular arc graph*. If, in addition, the family of arcs satisfies the Helly property (*i.e.*, if several arcs mutually intersect, then the intersection of these arcs is nonempty), G is called a *Helly circular arc graph* (a.k.a. Θ circular arc graph). G is called an *interval graph* if S is a family of intervals on a real line.

In this paper, we often use a pair of the two endpoints of an arc to denote a closed arc. If we move along an arc in the clockwise direction, the last point on the arc is *clockwise endpoint*. The other endpoint is *counterclockwise endpoint*. If we use $[l_0, l_1]$ to denote an arc, l_1 and l_0 represent clockwise and counterclockwise endpoints, respectively.

The aforementioned classes of graphs can also be defined as intersection graphs on some discrete objects. Take the circular arc graphs for example. Let D be a circularly ordered set (such as points on a circle). A circular arc of D is defined as any set of contiguous elements in D. Let S be a set of circular arcs on D. The intersection graph G of S is a circular arc graph and the pair (D, S) is a circular arc representation. Two IRs (D_1, S_1) and (D_2, S_2) are said to be *equivalent* if there exists a one-to-one onto function f: $S_1 \rightarrow S_2$ such that x and y in S_1 intersect iff $f(x)$ and $f(y)$ in S_2 intersect. An IR (D, S) is said to be *minimal* if there does not exist an equivalent IR (D', S'), where $D' \subset D$. An IR (D, S) is said to be *minimum* if, for any other equivalent IR (D', S'), $|D'| \geq |D|$. We call $|D|$ the *size* of the IR (D, S). An element, say d, in D is called an *intersection point* (IP) if there exist two elements (not necessarily distinct), say s_1 and s_2, in S such that $s_1 \cap s_2 = \{d\}$. An IR, say (D, S), is often denoted by a $|S| \times |D|$ $(0, 1)$-matrix. A row, say R, of the matrix corresponds to an element in S. $R(i) = 1$ iff the ith element of D is contained in the element of S. We simply refer to the matrix as an IR if no confusion arises.

Suppose $D_1 = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$, and $S_1 = \{\{\clubsuit, \diamondsuit\}, \{\diamondsuit, \heartsuit\}, \{\heartsuit, \spadesuit\}, \{\spadesuit\}\}$. The corresponding matrix is M_1. Let $D_2 = \{\mathsf{J}, \mathsf{Q}, \mathsf{K}, \mathsf{A}\}$, and $S_2 = \{\{\mathsf{J}\}, \{\mathsf{J}, \mathsf{Q}, \mathsf{K}\}, \{\mathsf{Q}, \mathsf{K}, \mathsf{A}\}, \{\mathsf{A}\}\}$. Then the corresponding matrix is M_2. It is easy to verify these two interval representations are equivalent. The minimal interval representation is M_3, which can be obtained by deleting a column from M_1 or M_2.

$$M_1 = \begin{bmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,1 \\ 0\,0\,0\,1 \end{bmatrix}, M_2 = \begin{bmatrix} 1\,0\,0\,0 \\ 1\,1\,1\,0 \\ 0\,1\,1\,1 \\ 0\,0\,0\,1 \end{bmatrix}, M_3 = \begin{bmatrix} 1\,0\,0 \\ 1\,1\,0 \\ 0\,1\,1 \\ 0\,0\,1 \end{bmatrix}$$

The computation models employed in this paper are more or less standard. One model used is the well known PRAM. Some of our algorithms are implemented on EREW PRAM. Some other algorithms are designed for the Common CRCW PRAM. Also mentioned is a stronger submodel called Priority CRCW PRAM. A more powerful model known as Broadcasting with Selective Reduction (BSR) introduced in Akl and Guenther [2] is also used here.

A PRAM algorithm is said to be *work-optimal* if the processor-time product matches the time lower bound of the sequential algorithm. We say an algorithm is *time-optimal* if its time bound matches the lower bound on the corresponding model.

In designing PRAM algorithms, we often use the following result, usually attributed to Brent [5], to obtain the best tradeoff between the time and processor bounds.

Theorem 1. *If a problem can be solved in $O(T)$ time with $O(P)$ PRAM processors, the problem can also be solved in $O(TP/P')$ time with $O(P')$ PRAM processors, for $P' < P$.*

Cook, Dwork and Reischuk [12] established the following lower bound.

Theorem 2. *Computing the OR of n bits requires at least $\Omega(\log n)$ time on exclusive-write machines.*

A lower bound on CRCW PRAM is given in Beame and Hastad [3].

Theorem 3. *Checking parity for n bits requires at least $\Omega(\log n/\log\log n)$ time on a Priority CRCW PRAM if a polynomially bounded number of processors are used.*

We say a problem is in NC if there is an algorithm for it that runs in polylog time with a polynomial number of processors. Such an algorithm is called NC algorithm. Any parallel algorithm mentioned in the paper is meant to be an NC algorithm.

One very useful procedure in parallel computing is the prefix computation. It is not hard to see the above PRAM lower bounds apply to the prefix computation. It is known that all prefix sums for an array of n elements can be obtained in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors [17], or in $O(\log n/\log\log n)$ time using $O(n \log\log n/\log n)$ Common CRCW PRAM processors [11], or in $O(1)$ time using $O(n)$ BSR processors [18].

One problem discussed in Chen [8] is the subarray computation. Given an array, say $a[1:n]$, composed of two types of elements, the problem is to obtain a subarray $b[1:k]$ of $a[1:n]$ such that $b[j]$ is the jth element in a of type 1, for $0 < j \leq k$, where k is the number of elements in a of type 1. The problem can be solved using the procedure for computing the prefix sums. It is now easy to conclude the following.

Theorem 4. *The subarray computation can be done in $O(\log n)$ time with $O(n/\log n)$ EREW PRAM processors, or in $O(\log n/\log\log n)$ time with $O(n\log\log n/\log n)$ Common CRCW PRAM processors. Both procedures are work-optimal. On the BSR model, the problem can be solved in $O(1)$ time with $O(n)$ processors. The algorithms are all time-optimal.*

These results are used frequently in designing other parallel algorithms. We may not make explicit reference to these results every time we use them later in this paper. In this paper, the minimal IR matrices are obtained thru column deletion. So the lower bound for subarray computation also applies to computing minimal representations.

2 Properties

In this section, we present some properties of minimal interval and circular arc representations. The proofs, if not given here, can be found in our earlier works [6] [10].

Lemma 5. *Suppose (D, I) is an interval representation. An element $d \in D$ is an IP iff there exist intervals I_i and $I_j \in I$ (I_i and I_j may be the same interval) such that d is the left endpoint of I_i and the right endpoint of I_j.*

Lemma 6. *Suppose (D, I) is an interval representation, and M is the corresponding matrix. An element of D is an IP iff the corresponding column of M is not contained in any other column.*

Theorem 7. *Suppose (D, I) is an interval representation and M is the corresponding matrix. The following four assertions are equivalent:*

1. *(D, I) is a minimum interval representation.*
2. *(D, I) is a minimal interval representation.*
3. *Every element of D is an IP.*
4. *No column of M contains another.*

Note the analogous statements of Theorem 7, Lemma 5 and Lemma 6 for circular arc representations are not true,. We will instead give the following results for the circular arc representation.

Lemma 8. *Suppose (D, A) is a circular arc representation, and M is the corresponding matrix. A column of M is not contained in any other column if the corresponding element of D is an IP.*

Theorem 9. *Suppose (D, A) is a circular arc representation and M is the corresponding matrix. For the following four assertions, one implies the next:*

1. *(D, A) is a minimum circular arc representation.*
2. *(D, A) is a minimal circular arc representation.*
3. *Every element of D is an IP.*
4. *No column of M contains another.*

Proof. ($1 \Longrightarrow 2$) By definition.
($2 \Longrightarrow 3$) Analogous to the corresponding part of the proof of Theorem 7.
($3 \Longrightarrow 4$) Immediate from Lemma 8. □

Theorem 10. *Suppose (D, A) is a circular arc representation. The representation is minimal iff every element of D is an IP.*

However, for circular arc representation satisfying the Helly property, the analogous statement of Theorem 7 is true. The proof is also analogous. We only list the result below.

Theorem 11. *Suppose (D, A) is a Θ circular arc representation and M is the corresponding matrix. The following four assertions are equivalent:*

1. *(D, A) is a minimum Θ circular arc representation.*
2. *(D, A) is a minimal Θ circular arc representation.*
3. *Every element of D is an IP.*
4. *No column of M contains another.*

The following theorem tells us something about the relation between minimal Θ circular arc representations and minimal circular arc representations.

Theorem 12. *If (D, A) is a minimal Θ circular arc representation for G, (D, A) is also a minimal circular arc representation for G.*

Proof. If (D, A) is a minimal Θ circular arc representation for G, then every element of D is an IP, by Theorem 11. Now the result follows from Theorem 10.
□

3 Algorithms

In this section we give concrete computational procedures for the recognition and construction of minimum interval and circular arc representations. We begin with the problem of testing for minimum interval representations. We note that if an $s \times t$ matrix is a minimum interval representation matrix, then $s \geq t$. The reason is, if the matrix is a minimum interval representation, then all columns of the matrix correspond to IPs by Theorem 7. Thus, the number of left endpoints and therefore the number of rows is at least t. Consequently, if $s < t$, we can conclude immediately the representation is not minimum. Suppose we have an interval representation matrix with size $s \times t$ $(s \geq t)$. We first check which columns correspond to the IPs. Obtaining the IPs is easy. By Lemma 5, a column corresponds to an IP iff it corresponds to a left endpoint and a right endpoint. For each row, we can decide its two endpoints in $O(1)$ time with t EREW PRAM processors. It then follows easily that all the IPs can be identified in $O(1)$ time with $O(st)$ Common CRCW PRAM processors. On the EREW PRAM model, the problem cannot be solved in $O(1)$ time. In fact, we have established a lower bound stated in the following theorem.

Theorem 13. *Deciding if an $s \times t$ interval representation matrix is minimum requires at least $\Omega(\log s)$ time on a CREW PRAM, for $s \geq t$.*

Proof. We prove the theorem by a reduction from computing the OR. Let $b[1]$, $b[2], \ldots, b[n]$ be n bits. We construct an $n \times n$ matrix M as follows.

$$M[i,j] = \begin{cases} 1 \text{ if } b[i] = 1 \wedge 0 < j \leq n \\ 0 \text{ if } b[i] = 0 \wedge i \neq j \wedge 0 < j \leq n \\ 1 \text{ if } i = j \end{cases}$$

Obviously, M can be constructed in constant time with $O(n^2)$ processors. According to the construction, all columns correspond to IPs iff all bits of b are 0, or equivalently, iff the OR of the n bits is 0. By Theorem 7, all columns of M correspond to IPs iff M is a minimum interval representation. Now the bound follows easily from Theorem 2. □

The $\Omega(\log s)$ lower bound also applies to the problem of deciding if an $s \times t$ (Θ) circular arc representation is minimal.

Below we show deciding if an interval representation is minimum can be done in $O(\log s)$ time by an optimal EREW PRAM procedure. The following is a procedure that computes the left endpoints.

```
for $i := 1$ to $t$ codo $l[i] := 0$ odoc; {initialize $l$}
for $i := 1$ to $s$ codo {initialize $ml$}
  for $j := 1$ to $t$ codo
    $ml[i,j] := 0$;
  odoc;
odoc;
for $i := 1$ to $s$ codo $ml[i, b[i]] := 1$ odoc;
for $i := 1$ to $t$ codo $l[i] := \bigvee_{j=1}^{s} ml[j,i]$ odoc;
```

After executing the above code, $l[i] = 1$ iff Column i corresponds to a left endpoint. The only step that requires more than constant time is Line 7, which can be done in $O(\log s)$ time on an EREW PRAM. All steps can be done optimally. In an analogous way, we can also decide which columns correspond to right endpoints. So we can now conclude the following.

Theorem 14. *Given an $s \times t$ interval representation matrix, deciding if the representation is minimum can be done in $O(\log s)$ time with $O(st/\log s)$ EREW PRAM processors, or in $O(1)$ time with $O(st)$ Common CRCW PRAM processors. Both algorithms are time-and-work-optimal.*

If an interval representation matrix is not minimum, we can obtain a minimum one by deleting some columns. However, we cannot obtain such a matrix by simply deleting all columns that do not correspond to IPs initially, since one column may become to correspond to an IP as a result of deleting a neighboring column. Below we describe a sequential procedure for obtaining a minimum interval representation.

First compute $l[i]$ and $r[i]$ for $0 < i \leq t$. Then perform the following task.

```
0  for $i := 1$ to $t$ do $m[i] := l[i] \wedge r[i]$ od;
1  $i := 1$;
2  while $i <= t$ do
3    if $l[i] = 1 \wedge r[i] = 0$ then
4      while $r[i] = 0$ do $i := i + 1$ od;
5      $m[i] := 1$;
6    fi;
7    $i := i + 1$;
8  od;
```

Line 0 sets $m[i]$ to 1 iff Column i corresponds to an IP. Then we scan the columns from left to right (Lines 2–8). If a column corresponds to a left endpoint but not a right endpoint, we will repeatedly remove columns (keep $m[i]$ as 0) until we have reached a column that corresponds to a right endpoint (Line 4). Then the column corresponds to an IP and will remain (set $m[i]$ to 1 at Line 5). When the above procedure terminates, the columns that correspond to IPs ($m[i] = 1$) form a minimum interval representation.

The parallel procedure can work as follows. First, set $m[i]$'s as Line 0. Then, for each left endpoint that does not correspond to an IP, find the closest right endpoint, say k, to its right, and set $m[k]$ to 1. This can be easily done within the resource bounds for parallel prefix computation. Once we have identified all the columns corresponding to maximal cliques, we simply apply subarray computation on all the rows. It is now easy to conclude the following.

Theorem 15. *Given an $s \times t$ interval representation matrix, a minimum interval representation can be obtained in $O(\log(st))$ time with $O(st/\log(st))$ processors by a time-and-work-optimal EREW PRAM algorithm, or in $O(\log t/\log\log t)$ time with $O(st \log\log t/\log t)$ processors by a time-and-work-optimal Common CRCW*

PRAM algorithm, or in $O(1)$ time with $O(st)$ processors by a time-optimal BSR algorithm.

Next we consider the problem of deciding if a circular arc representation is minimal. If follows from Theorem 10 that the problem can be solved by checking if each column of the representation matrix corresponds to an IP. For the same reason as above, if the input matrix is of size $s \times t$ and $s < t$, we can conclude immediately the representation is not minimal. So we only need to consider the case $s \geq t$.

The procedure works as follows. We first locate all the clockwise and counterclockwise endpoints. Then for each column, say i, perform the following task. If the column corresponds to both clockwise and counterclockwise endpoints, then find the shortest arcs, say a and b, whose clockwise and counterclockwise endpoints are i, respectively. Then i is an IP iff the size of intersection of a and b is 1. Both a and b can be found using a variation of the procedure for finding the first 1 in a (0,1)-array, which takes $O(1)$ time and $O(t)$ Common CRCW PRAM processors [13], or $O(\log t)$ time and $O(t/\log t)$ EREW PRAM processors. It is now easy to conclude the following.

Theorem 16. *Given an $s \times t$ circular arc representation matrix, deciding if the representation is minimal can be done in $O(\log s)$ time with $O(st/\log s)$ EREW PRAM processors, or in $O(1)$ time with $O(st)$ Common CRCW PRAM processors. Both algorithms are time-and-work-optimal.*

Below we consider how to construct a minimal circular arc representation. The method is sketched as follows. First we obtain all the clockwise endpoints $e[i]$'s and counterclockwise endpoints $b[i]$'s. Then starting from the first column, we perform the following task for each column: Check, based on the values of $b[i]$'s and $e[i]$'s, if the current column, say c, corresponds to an IP. If so, set $m[c]$ to 1. Otherwise, delete the column (keep $m[c]$ as 0) and update $b[i]$'s and $e[i]$'s if applicable. At the end of the iteration, each remaining column corresponds to an IP, and all the remaining columns form a minimal circular arc representation, by Theorem 10.

We are now going to present an efficient implementation of the algorithm. For the convenience of the description, we assume, in the following, that the indices of the first row and the first column are both 0. We also assume, without loss of generality, that the input matrix does not contain an all-1 row.

```
for $i := 0$ to $s - 1$ do {compute $b$ and $e$}
  for $j := 0$ to $t - 1$ do
    if $M[i,j] = 1 \wedge M[i,(j+1) \bmod t] = 0$ then $e[i] := j$ fi;
    if $M[i,j] = 1 \wedge M[i,(j+t-1) \bmod t] = 0$ then $b[i] := j$ fi;
  od;
od;
for $i := 0$ to $t - 1$ do $m[i] := 0$ od; {initialize $m$}
for $i := 0$ to $t - 1$
  find shortest arc, say $j$, whose clockwise endpoint is $i$;
  find shortest arc, say $k$, whose counterclockwise endpoint is $i$;
```

```
10      if both $j$ and $k$ exist and size of their intersection is 1 then $m[i] := 1$ fi;
11      if $m[i] = 0$ then {delete Column $i$}
12        for $j := 0$ to $s - 1$ do {update $b$ and $e$, if applicable}
13          if $b[j] = i$ then $b[j] := (i + 1) \bmod s$ fi;
14          if $e[j] = i$ then $e[j] := (i - 1 + s) \bmod s$ fi;
15        od;
16      fi;
17    od;
```

Computing the $b[i]$'s and $e[i]$'s (Lines 0–5) is straightforward and takes $O(st)$ time. Lines 8–9 can be easily done in $O(t)$ time. If both j and k exist, we have two arcs $[b[j], i]$ and $[i, e[k]]$. The size of their intersection can be decided in constant time. So we can now conclude the following.

Theorem 17. *Given an $s \times t$ circular arc representation matrix, a minimal circular arc representation can be obtained in $O(st)$ time.*

Suppose M_4 is the input circular arc representation. None of the columns correspond to any IP. When Columns 0 and 1 have been deleted, Column 2 becomes to correspond to an IP. So Column 2 is not removed, according to the above procedure. Then the rest of the columns are all deleted. So the minimal circular arc representation matrix is a column that consists of six 1's only.

$$M_4 = \begin{bmatrix} 1&1&1&0&0&1 \\ 0&1&1&1&0&0 \\ 0&0&1&1&1&1 \\ 0&1&1&1&1&0 \\ 1&1&1&1&0&0 \\ 1&1&1&1&1&0 \end{bmatrix}, \quad M_5 = \begin{bmatrix} 1&1&0 \\ 0&1&0 \\ 0&1&1 \\ 0&1&1 \\ 1&1&0 \\ 1&1&1 \end{bmatrix}$$

To obtain an efficient parallel algorithm, we make some observations. It is easy to see that deleting a column that does not correspond to an IP yields an equivalent circular arc representation. Moreover, being an IP is not affected by deleting some columns. Nevertheless, a column can change status and become to correspond to an IP as a result of deleting another column, even though the two columns are not next to each other. Suppose we have two arcs $[i, j]$ and $[j, i]$. If Column i is deleted, Column j becomes to correspond to an IP and cannot be deleted. We say two arcs *embrace* if they intersect at both endpoints but neither is contained in the other. If we delete a column, the two neighboring columns and any embracing columns may change status. It should be obvious that several columns can be deleted simultaneously if the deletion of one column does not affect the status of another column. So, if we can identify those columns efficiently, we can also obtain a more compact circular arc representation efficiently.

```
0   compute $b$ and $e$ in parallel;
1   for $i := 0$ to $t - 1$ codo $m[i] := 0$ odoc; {initialize $m$}
2   $min := 0$; $n := t$;
3   while $min = 0$ do {consider odd columns}
```

```
4     set $lend[i,j]$ and $rend[i,j]$ to 0, for $0 \le i,j < n$;
5     for $i := 1$ step 2 to $n-1$ codo {compute lend and rend}
6       for $j := 0$ to $s-1$ codo
7         if $e[j] = i$ then $lend[b[j],i] := 1$ fi; {$lend[i,j] = 1$ means $[i,j]$ is an arc}
8         if $b[j] = i$ then $rend[e[j],i] := 1$ fi; {$rend[i,j] = 1$ means $[j,i]$ is an arc}
9       odoc;
10      if $i$ is an IP then $m[i] := 1$ fi;
11    odoc;
12    construct $G = (V,E)$, where $V = \{v_i \mid i \bmod 2 = 1 \wedge m[i] = 0 \wedge 0 < i < n\}$
13      and $E = \{(v_i, v_j) \mid lend[i,j] = rend[i,j] = 1\}$;
14    find a maximal independent set $V'$ of $G$;
15    $m[i] := 1$ for all $v_i \in V - V'$;
16    delete Column $i$ of $M$ and $m[i]$ if $i \bmod 2 = 1$ and $m[i] = 0$, for $0 < i < n$.
17    let $n'$ be the number of columns in the resulting matrix.
18    if $n' = n$ then $min := 1$ else update $b$ and $e$ fi;
19    $n := n'$;
20  od; {next consider even columns}
21  set $lend[i,j]$ and $rend[i,j]$ to 0, for $0 \le i,j < n$;
22  for $i := 0$ step 2 to $n-1$ codo {compute lend and rend}
23    for $j := 0$ to $s-1$ codo
24      if $e[j] = i$ then $lend[b[j],i] := 1$ fi;
25      if $b[j] = i$ then $rend[e[j],i] := 1$ fi;
26    odoc;
27    if $i$ is an IP then $m[i] := 1$ fi;
28  odoc;
29  construct $G = (V,E)$, where $V = \{v_i \mid i \bmod 2 = 0 \wedge m[i] = 0 \wedge 0 \le i < n-1\}$
30    and $E = \{(v_i, v_j) \mid lend[i,j] = rend[i,j] = 1\}$;
31  find a maximal independent set $V'$ of $G$;
32  $m[i] := 1$ for all $v_i \in V - V'$;
33  delete Column $i$ of $M$ and $m[i]$ if $i \bmod 2 = 0$ and $m[i] = 0$, for $0 \le i < n-1$.
34  if the last column does not correspond to an IP then remove it fi;
```

In order to achieve a polylog time bound, we must identify columns that can be deleted simultaneously. We first consider odd columns (columns whose indices are odd numbers) that do not correspond to IPs. Obviously, none of them are neighbors. However, we may not remove all those columns since there may be some embracing arcs. To resolve this problem, we construct a graph (Lines 12–13) as follows. Associate each such column (endpoint) with a vertex. If two arcs embrace and the size of their intersection is 2, then link the two vertices corresponding to the two endpoints. So, if two vertices are adjacent in the resulting graph, only one of the two columns can be deleted. If several vertices are mutually independent , then all of the columns can be deleted simultaneously. Therefore, we compute a maximal independent set of the graph (Line 14). Because of the maximality, any vertex not in the independent set is adjacent to a vertex in the independent set. Consequently, the columns associated with the vertices outside the independent set become to correspond to IPs when the columns associated with the vertices inside the independent set have been deleted. After the deletion of some columns, some even columns may become odd columns. So we repeat this process and consider the odd columns again until all odd columns correspond to IPs (Lines 3–20).

Then, in an analogous way, we consider even columns among Columns from 0 to $(n-2)$ (we do not consider Column $(n-1)$ for the time being since Columns 0 and $(n-1)$ are neighbors) that do not correspond to IPs. However, in this case, after some columns are deleted, we do not need to repeat the process since all columns except possibly the last one now correspond to IPs. Finally, we check if the last column corresponds to an IP. If not, simply delete it. It is now easy to conclude the procedure correctly constructs a minimal circular arc representation, by Theorem 10.

If the procedure runs on M_4, Lines 12–13 construct a graph with three vertices corresponding to Columns 1, 3, and 5, respectively. The graph does not contain any edge. So all three columns are removed at Line 16 and the intermediate matrix is M_5. Then Column 1 corresponds to an IP and the **while** loop cannot remove any more column, so the control goes on to Line 21 and we consider even columns. The first column of M_5 is removed at the end of Line 33. And finally Line 34 deletes the last column.

Before deriving efficient resource bounds for the above procedure, we make some additional assumptions. The first one is $s = O(t^2)$. If this is not true, the input matrix contains some identical rows. In this case, we can simply remove and make row duplicates at the beginning and at the end, respectively. We claim this can be done within $O(\log(st))$ time and $O(st)$ work. Obviously, any row, say i, can be represented by a pair, $(b[i], e[i])$, where $0 \leq i < s$ and $0 \leq b[i], e[i] < t$. First we sort the rows using radix sort. Then we can identify row duplication in constant time with $O(s)$ EREW PRAM processors. We sort $e[i]$'s by actually sorting $e'[i]$'s, where $e'[i] = se[i] + i$. So each $e'[i]$ is distinct and is in the range $[0, st-1]$. Such numbers can be sorted in $O(\log(st))$ time with $O(st/\log(st))$ EREW PRAM processors [7]. Then $b[i]$'s can be sorted using the same method. Now, the validity of the claim follows immediately.

The next assumption is $t = O(s)$. If this is not true, some columns do not correspond to any endpoint. In this case, we simply delete those columns. This can obviously be done in $O(\log(st))$ time and $O(st)$ work on an EREW PRAM.

With the above assumptions, let's now consider the resource requirements of the procedure. It is easy to see the most expensive part is the **while** loop (Lines 3–20). Lines 5–11 can be done in $O(\log s)$ time with $O(st)$ work on EREW PRAM. One challenging step is to find a maximal independent set (Line 14). Currently, the most efficient algorithm solves the problem in $O(\log^3 v)$ time with $O((v+e)\log^2 v)$ work on an EREW PRAM [14], where v and e denote, respectively, the number of the vertices and the edges in a graph. In our case, $v \leq t/2 = O(t)$. Since each edge corresponds to two embracing arcs (rows), we have $e \leq s/2 = O(s)$. Therefore, Line 14 takes $O(\log^3 t)$ time with $O((s+t)\log^2 t)$ work on an EREW PRAM. The total work of the loop body (Lines 4–19) is $O(st + (s+t)\log^2 t) = O(st)$ since $t = O(s)$. The total time of the loop body is $O(\log s + \log^3 t)$. By Assumption 1, $\log s = O(\log t)$. So the loop body takes $O(\log^3 t)$ time. Since the loop iterates $O(\log t)$ times, the loop takes $O(\log^4 t)$ time and $O(st \log t)$ work. Now the total resource bounds follow easily.

Theorem 18. *Given an $s \times t$ circular arc representation matrix, a minimal circular arc representation can be obtained in $O(\log s + \log^4 t)$ time with $O(st \log t / (\log s + \log^4 t))$ processors on an EREW PRAM.*

The time lower bound on the EREW PRAM is $\Omega(\log(st))$ and does not match the above upper bound. We conjecture the algorithm is not time-optimal. The algorithm is work-optimal within a factor of $O(\log t)$. If the input is a Θ circular arc representation matrix, then the above algorithm is definitely not time-optimal. Below we describe a time-optimal procedure.

First, we eliminate all column duplication. Then, we check, for each column, if it is contained in another column. If so, delete it. It follows that, in the resulting matrix, no column contains another. So the resulting matrix gives a minimum Θ circular arc representation by Theorem 11, and also a minimal circular arc representation by Theorem 12. Deciding whether Column i contains Column j takes constant time with $O(s)$ Common CRCW PRAM processors. There are $O(t^2)$ pairs of columns. So all the IPs can be identified in constant time with $O(st^2)$ Common CRCW PRAM processors. On the EREW PRAM, this can be done in $O(\log(st))$ time with $O(st^2)$ work. Once all the IPs are identified, we can obtain a minimum Θ circular arc representation by applying subarray computation on all the rows. Now, it is easy to conclude the following.

Theorem 19. *Given an $s \times t$ Θ circular arc representation matrix, a minimum Θ circular arc representation (also a minimal circular arc representation) can be obtained in $O(\log(st))$ time with $O(st^2 / \log(st))$ EREW PRAM processors, or in $O(\log t / \log\log t)$ time with $O(st^2 \log\log t / \log t)$ Common CRCW PRAM processors, or in $O(1)$ time with $O(st^2)$ BSR processors. All algorithms are time-optimal.*

Although the above procedure for computing Θ circular arc representations is time-optimal, the total work involved moves away from optimality.

4 Discussion

In this paper, we have studied the properties of minimal interval and circular arc representations and have given some efficient algorithms for the recognition and construction of such representations. The models of parallel computation used in this paper include EREW PRAM, CRCW PRAM, and BSR. We have presented algorithms for each of these models and they are of independent interest. One might ask the following questions: Which model is the best? Is it sufficient to design algorithms on only one of the models? It is often debatable whether one model is better than another. There is no universal agreement on the answer. It seems premature to tell at this point. There is an interesting project on building PRAM-type computers (see, *e.g.*, [1]). BSR is a relatively new model; the analogous project is not known currently but it is technically feasible (see, *e.g.*, [18]). As in the way of deciding Gordon Bell Prize winners, probably a good way to compare the performance between PRAM and BSR computers is to run sample programs on both types of machines, in which case efficient algorithms on both PRAM and BSR are needed. Perhaps someday, the algorithms in this paper will also be used for this purpose.

References

1. F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, and D. Scheerer. On the physical design of PRAMs. *Computer Journal*, 36(8):756–762, December 1993.
2. S. G. Akl and G. R. Guenther. Broadcasting with selective reduction. In G. X. Ritter, editor, *Proceedings, 11th IFIP World Computer Congress*, pages 515–520, 1989. North-Holland.
3. P. W. Beame and J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*, 36(3):643–670, July 1989.
4. S. Benzer. On the topology of the genetic fine structure. *Proceedings, Nat. Acad. Sci.*, 45:1607–1620, 1959.
5. R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21:201–208, 1974.
6. L. Chen. Efficient parallel algorithms for several intersection graphs. In *Proceedings, 22nd Int'l Symp. on Circuits and Systems*, pages 973–976. IEEE, 1989.
7. L. Chen. Efficient deterministic parallel algorithms for integer sorting. In S. G. Akl, F. Fiala, and W. W. Koczkodaj, editors, *Proc. International Conference on Computing and Information, Lecture Notes in Computer Science, Vol. 468*, pages 433–442. Springer-Verlag, 1990.
8. L. Chen. Optimal parallel time bounds for the maximum clique problem on intervals. *Information Processing Letters*, 42(4):197–201, June 1992.
9. L. Chen. Efficient parallel recognition of some circular arc graphs, I. *Algorithmica*, 9(3):217–238, March 1993.
10. L. Chen. Revisiting circular arc graphs. In D.-Z. Du and X.-S. Zhang, editors, *Proceedings, 5th Annual International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 834*, pages 559–566. Springer-Verlag, 1994.
11. R. Cole and U. Vishkin. Faster optimal parallel prefix sums and list ranking. *Information and Computation*, 81(3):334–352, June 1989.
12. S. A. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15(1):87–97, 1986.
13. F. E. Fich, P. L. Ragde, and A. Wigderson. Relations between concurrent-write models of parallel computation. In *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, pages 179–189. Association for Computing Machinery, 1984.
14. M. Goldberg and T. Spencer. Constructing a maximal independent set in parallel. *SIAM J. on Discr. Math.*, 2(3):322–328, August 1989.
15. U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.
16. J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990.
17. R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.
18. L. F. Lindon and S. G. Akl. An optimal implementation of broadcasting with selective reduction. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):256–269, March 1993.