THE TELEPHONE CONNECTION PROBLEM

Vijaya Ramachandran and Li-Chung Wang

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-16 April 1991

The Telephone Connection Problem [1]

Vijaya Ramachandran[2] and Li-Chung Wang[3]

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

The telephone connection problem (TCP) is the problem of simulating a telephone link of fixed capacity to assess its ability to serve incoming calls. This simulation is performed on a large number of sample calls at AT&T Bell Laboratories. In order to speed up the simulation, it is desirable to obtain good parallel algorithms for the problem.

We first show that a natural decision problem that is related to the TCP is in NC. Then, we present two parallel algorithms for the telephone connection problem. We give an $O(k \log n)$ time parallel algorithm on an EREW PRAM for the TCP using n processors, where k is the capacity of the telephone line and n is the number of calls. Then, we improve our algorithm to run in $O(\min(\sqrt{n}, k) \log n)$ time on an EREW PRAM using n processors. Finally, we prove that the TCP is a CC-complete problem. The CC-completeness result for the TCP does not automatically imply an $O(\sqrt{n}polylog(n))$ time algorithm for the problem, since the reduction we use results in a quadratic blow-up in size.

[1] This work was supported in part by NSF grant CCR-89-10707.

[2] E-mail address: vlr@cs.utexas.edu

[3] E-mail address: lcwang@cs.utexas.edu

1 Introduction

In this paper, we consider the telephone connection problem (TCP) [2]. Informally, an instance of the TCP consists of a link of fixed capacity and a sequence of calls specified by their starting and finishing times. A call can be served if the link is not full to capacity. A call that cannot be served at its starting time is discarded. The TCP problem is to decide which calls can be served.

The TCP problem has a simple sequential algorithm but not much was known about its parallel complexity. It is of some importance to obtain good parallel algorithms for the problem since at AT&T Bell Laboratories TCP inputs of very large size are solved routinely. The researchers there are interested in speeding up the simulation by running it on a parallel machine [2].

We first show that given an instance of the TCP, to decide if all calls can be served is in NC. Then, we present two parallel algorithms for the TCP problem. The first algorithm solves the TCP in time $O(k \log n)$ using n processors, where k is the number of available channels on the telephone line and n is the number of calls. We then improve the algorithm to bound its run time by $O(\min(k, \sqrt{n}) \log n)$ using n processors. Finally, we show that the comparator circuit value (C-CV) problem is log space reducible to the TCP and the TCP is log space reducible to the C-CV. Hence, the TCP is CC-complete [4] [7] when k is arbitrary. Currently, CC-complete problems are not known to be in NC or to be P-complete.

Our $O(\min(k, \sqrt{n}) \log n)$ time parallel algorithm for the problem uses some novel techniques, and introduces a new prefix sums computation that we call the 'subrange prefix sums'. The CC-completeness result for the TCP does not automatically imply an $O(\sqrt{n}polylog(n))$ time algorithm for the problem, since our reduction from the TCP to the C-CV problem results in a quadratic blow-up in size. On the other hand, our results imply an $O(\min(k, k', \sqrt{n}) \log n)$ time parallel algorithm using n processors for the C-CV problem, where k represents the number of inputs for the circuit with value 1, k' represents the number of inputs for the circuit with value 0, and n is the number of gates. The previous best parallel algorithm for the C-CV problem runs in $O(\sqrt{n}polylog)$ time [7].

The parallel computation model we use here is the EREW PRAM. For PRAM models and techniques for designing efficient parallel algorithms on a PRAM model, see [5].

2 Problem Definition and an Algorithm

Given a telephone line with a fixed capacity, and a sequence of calls, the telephone connection problem (TCP) is to decide which calls can be served. Formally, let $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ represent the sequence of calls, where s_i represents the starting time of the ith call and f_i represents the finishing time of the ith call. Let k be the capacity of the telephone line, i.e. the total number of available channels. The ith call can be served if at time s_i, the number of calls being served is less than k; otherwise, the ith call should be discarded.

Definition 1 Given the sequence $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ and the integer k as above, the **telephone connection problem (TCP)** is to decide which calls can be served. Let $i_1, i_2, \ldots, i_j$ be the indices of calls which can be served, then $\{i_1, i_2, \ldots, i_j\}$ is the **solution set** of the problem.

For convenience, we assume $s_i \neq s_j$ for all $i \neq j$; otherwise we will need a tie-breaking procedure to decide which call should be served first. For $s_i = f_j$, $i \neq j$, we assume that the ith call cannot use the same

slot used by call j because slot release and slot allocation require some amount of time. Thus, if $s_i = f_j$, we can simply increase the value of f_j by a small amount to avoid conflict. Hence, we assume that $s_i \neq f_j$ for all i,j. We also assume for convenience that $s_1 < s_2 < \ldots < s_n$.

There is a simple sequential algorithm to solve the TCP. We can scan the sequence of calls by their order of starting times, and a call will be served if at its starting time, the telephone line is not full. In this paper, we consider the parallel complexity of the TCP.

We first consider a natural decision problem that is related to the TCP. The problem is to decide if all calls can be served. This is equivalent to deciding if there exists a call that cannot be served. To solve this problem, we assume that all calls can be served and then for each call i we compute the number of calls that occupy the telephone link at time s_i. Given a sequence of n calls specified by their starting times and finishing times, we first sort the timing values in nondecreasing order to obtain a sorted array S of $2n$ values. Then, we create a new array A such that $A(i) = 1$ if $S(i)$ contains a starting time value and $A(i) = -1$ otherwise, and perform the prefix sums computation on array A. Let B be the resulting array of the prefix sums. $B(i)$ will contain the number of calls that occupy the telephone link at time t, where $S(i) = t$, assuming that all calls can be served. Hence, by finding the first entry in B with value greater than the capacity of the telephone line, we can determine the first call that cannot be served if it exists. Since all these steps can be done in $O(\log n)$ time using n processors, this decision problem is in NC.

By the above result, the TCP problem for those input instances that contain a *polylog* number of calls that cannot be served is also in NC. The algorithm runs for a *polylog* number of iterations and during each iteration, finds the first call that cannot be served and removes it. Then, at the end of iteration i, the algorithm can identify the first i calls that cannot be served.

In the following, we consider the general case of the TCP problem that could contain a large number of calls that cannot be served. We present an algorithm for the TCP which runs in time $O(k \log n)$ using n processors, where k is the capacity of the telephone line and n is the number of calls. We have been informed [3] that a similar algorithm has been obtained independently by D. Nicol, B. Lubachevsky, and A. Greenberg.

Definition 2 Let $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ be the starting and finishing times in an input to the TCP. Let $G = (V, E)$ be a directed acyclic graph such that $V = \{1, 2, \ldots, n+1\}$, and if $s_{j-1} < f_i < s_j$, where $s_{n+1} = \infty$, then $(i, j) \in E$. Then, we call G the **telephone problem graph (TPG)** of the original input to the TCP. Let S be the solution set for an instance of TCP with input capacity k, and let G be its TPG. Then, we call S the **solution set of G with capacity k**.

Figure 1 gives an example of a TCP and its TPG. The following algorithm converts an input to the TCP into its TPG.

Algorithm A Construction of the telephone problem graph.

Input: $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ such that $s_1 < s_2 < \ldots < s_n$.

Output: TPG $G = (V, E)$ represented by an array $G(1 \cdots n+1)$ such that $(i, j) \in E$ iff $G(i) = j$.

G :

1	2	3	4	5	6
5	4	6	5	6	6

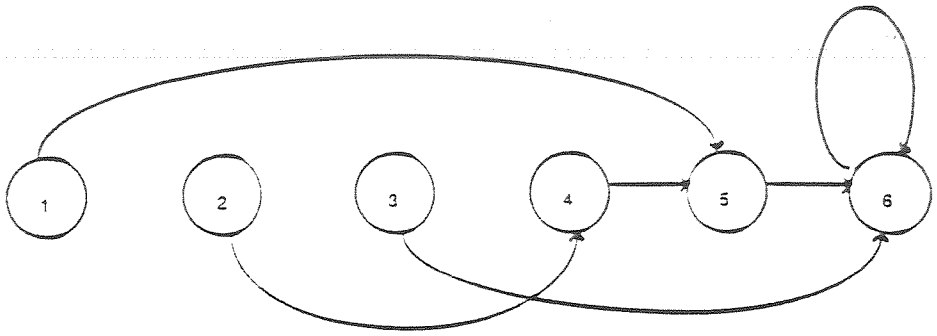

Figure 1:
An example of a TPG, corresponding to the starting and finishing times:
$(s_1, f_1) = (2.3, 10.4), (s_2, f_2) = (3.4, 7.2), (s_3, f_3) = (5.7, 23), (s_4, f_4) = (10, 10.9), (s_5, f_5) = (11.9, 22)$.

1. Sort the timing values s_i, f_i in nondecreasing order to obtain a sorted array S of $2n$ values.
2. Construct the following array L with length $2n$.
   If $S(i) = s_j$ for some $1 \leq j \leq n$ then $L(i) := 1$
   else $L(i) := 0$.
3. Compute the prefix sums on L and store the result back in L.
4. $\forall i, 1 \leq i \leq 2n$, do in parallel if $S(i) = f_j$ for some j then $G(j) := L(i) + 1$.
5. $G(n+1) := n + 1$.

End.

Step 1 can be implemented in time $O(\log n)$ using n processors by parallel merge sort [1]. The complexity of step 3 is $O(\log n)$ using $O(n/\log n)$ processors [6]. Thus, algorithm A runs in $O(\log n)$ time using n processors. In what follows, we prove the correctness of algorithm A.

The following observation follows from the definition of TPG.

Observation 1 *Let G be a TPG. then for each vertex i, there is exactly one outgoing arc.*

Lemma 1 *Algorithm A correctly converts an input to the TCP into its TPG.*

Proof. By observation 1, TPG G can be represented by an array $G(1 \cdot \cdot n+1)$. In step 4, $S(i) = f_j, L(i) = l$ implies that $\exists s_1, s_2, \ldots, s_l, s_{l+1}$ such that $s_1 < s_2 < \ldots < s_l < f_j < s_{l+1}$, so $G(j) = l + 1$ and $(j, l+1)$ is an arc in G. □

Lemma 2 *Let G be the TPG for an input to TCP and let (i, j) be an arc in G. If the ith call is served then the jth call will be served.*

Proof. If (i, j) is an arc in G, then $s_{j-1} < f_i < s_j$. If the ith call is served, this means that at time f_i, there will be a channel released by the ith call and that channel becomes available at time s_j. Therefore, the jth call can use that empty channel and will be served. □

Note that algorithm A provides a mapping from an instance of TCP to an instance of TPG. This mapping is a many-to-one function. Lemma 2 shows that we may be able to solve the TCP problem by working on its TPG. That is, if we assign label 1 to node i to mean that it can be served, and (i, j) is an arc in G, then we should also assign label 1 to node j since it can also be served. Algorithm B is based on this strategy.

In the following, instead of working on an instance of the TCP problem directly, we will work on a TPG. Thus, a node i will mean not only the node i in a TPG but also the ith call in the corresponding instance of the TCP.

Algorithm B Solution of the TCP.

Input: A TPG $G = (V, E)$ and an integer k.

Output: Node i will be labeled 1 iff $i \in$ the solution set of G on k.

$count := 0$

Repeat the following steps:

0. $count := count + 1$
1. Assign label 1 to the node with the smallest index. Let this node be v.
2. Assign label 1 to all nodes that are reachable from v, except the last dummy node.
3. $\forall$ node i, if $(i, j) \in E$, nodes $j, j+1, \ldots, j+m-1$ have been labeled 1 in step 2, and node $j+m$ has not been labeled 1 in step 2, then remove arc (i, j) and add arc $(i, j+m)$ to E.
4. Remove all the nodes that have been labeled 1, and their outgoing arcs to obtain a new TPG G.

Until either G contains only one vertex or $count = k$.

End.

Steps 2 and 3 of algorithm B can be implemented using pointer jumping. The remaining steps are straightforward. Thus, the time complexity for algorithm B is $O(k \log n)$ using n processors.

In what follows, we prove the correctness of algorithm B.

Lemma 3 *Let $G = (V, E)$ be a TPG. The set $S = \{i_1, i_2, \ldots, i_m\}$, where $s_{i_1} < \ldots < s_{i_m}$, is the solution set of G on $k = 1$ iff $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m) \in E$, the outgoing arc of i_m goes to the last node, and i_1 was assigned 1 in step 1 of algorithm B in the first iteration.*

Proof. ($\Rightarrow$) Since a call c whose starting time is between $s_{i_{(j-1)}}$ and s_{i_j} is not served, we know that the channel is occupied by call i_{j-1} from time $s_{i_{(j-1)}}$ to time t, where $s_{(i_j-1)} < t < s_{i_j}$. By the definition of the TPG, this implies the arc from i_{j-1} to i_j is in E.
($\Leftarrow$) By lemma 2. □

Definition 3 Let $G = (V, E)$ be the TPG for an input to the TCP. Let $\{i_1, i_2, \ldots, i_j\}$ be a set of nodes in G which are labeled 1. Construct $G' = (V', E')$ from G such that $V' = V - \{i_1, i_2, \ldots, i_j\}$, and $\forall i \in V'$, if $(i, l) \in E$ and $l, l+1, \ldots, l+m-1 \in \{i_1, \ldots, i_j\}$ but $l+m \notin \{i_1, \ldots, i_j\}$, then $(i, l+m) \in E'$. We call G' the **subproblem graph** of G.

Note that the new graph obtained at the end of step 4 of algorithm B is a subproblem graph with respect to the graph obtained from the previous iteration.

Theorem 1 *Algorithm B correctly compute the solution set of a TPG G with capacity k.*

Proof. We will prove the following assertion: At the end of the ith iteration of the Repeat loop, a call is labeled 1 *iff* it belongs to the solution set of G with capacity i. We will establish the assertion by induction on i.

Lemma 3 establishes the base case. Assume the assertion holds until $i-1$. Consider iteration i.

Let $G' = (V', E')$ be the subproblem graph obtained at the end of iteration $i-1$. Let $S = \{i_1, i_2, \ldots, i_j\}$ be the set of nodes that are labeled 1 during iteration i. We first show that S is contained in the solution set of G with capacity i by induction on the indices of subscripts of calls in S.

For the base case, consider i_1. Calls $1, 2, \ldots, i_1 - 1$ can be served by using $i-1$ channels, so channel i can be used to serve call i_1.

Assume the induction hypothesis holds until call i_{p-1}. Consider call i_p. $\exists i_q, i_q \in S, s_{i_q} < s_{i_p}$, $(i_q, i_p) \in E'$, and the call i_q was labeled 1. By the definition of G', this implies one of the following two cases:

(a). $(i_q, i_p) \in E$. By the induction hypothesis and lemma 2, i_p is in the solution set of G with capacity i.

(b). $(i_q, l) \in E$ and $l, l+1, \ldots, i_p - 2, i_p - 1$ were labeled 1 before entering iteration i. By the induction hypothesis the calls $l, l+1, \ldots, i_p - 1$ can be served using $i-1$ channels and call i_q can be served using i channels, so call i_p can be served on input G with capacity i.

Now consider a call j that is not labeled 1 at the end of iteration i. By the induction hypothesis, call j cannot be served using $i-1$ channels. Let $s_p < s_j < s_q$, where p, q are labeled 1 during iteration i and no other r, s that are labeled 1 during iteration i satisfy $s_p < s_r < s_j < s_s < s_q$. Since the arc (p, q) is in the subproblem graph obtained at the end of iteration $i-1$, we know that $s_j < f_p$. Hence, call j cannot be served on input G with capacity i. This concludes the proof of the assertion.

The assertion immediately implies the theorem. □

3 A Faster Algorithm for the TCP

In section 1, we presented an $O(k \log n)$ time algorithm using a linear number of processors for the TCP problem. When k is very large, say $\Theta(n)$, the algorithm becomes very inefficient. In this section, we give another algorithm which is similar to algorithm B but is better in the sense that no matter what k is, the run time is bounded by $O(\min(k, \sqrt{n}) \log n)$ using n processors.

Recall that in algorithm B, our strategy is to process one channel during one iteration, so during iteration i, we are processing channel i. However, we can speed up this process by allocating k channels to k calls initially and processing as many calls as we can during each iteration.

In what follows, we use an example to illustrate the ideas behind the faster algorithm. Consider the following TPG:

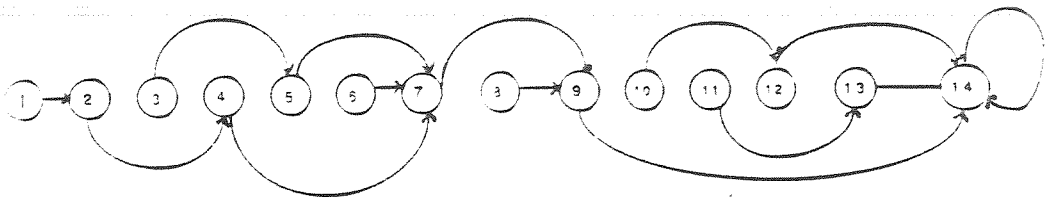

Figure 2: Another example of a TPG.

Let $k = 3$. Then, initially, we assign label 1 to nodes 1,3,6, which are the first three nodes with no incoming arc, and propagate 1's along the arcs. After we finish the propagation, the nodes 1,2,3,4,5,6,7,9 will be labeled 1. Note that node 14 has infinite starting time in the definition of TPG so we can ignore it. Observe that at time s_7, there will be three channels released by calls 4,5,6. One of these channels will be used by call 7. Since there is an arc $(7, 9)$, call 9 can use the same channel as call 7 used. Hence, the remaining two channels released by calls 4,5,6 can be used to serve calls 8 and 10.

Next, we start a new round of propagation with the initial condition that nodes 8 and 10 have label 1. These are the two calls that should be served by using two of the three channels released by calls 4,5,6, but did not receive label 1 during the first propagation phase. Notice that there is an arc $(8, 9)$, and recall that call 9 can use the same channel used by call 7, so the channel released by call 8 will become available at time s_{10}. However, we have just assigned a channel to call 10 by giving it label 1. Hence, call 11 can use the channel released by call 8. This implies that before we start the second round of propagation, we should remove all nodes that have been labeled 1 during the first round of propagation together with their outgoing arcs, and add an arc $(8, 11)$. This removal process is same as what we did in step 3 and 4 of algorithm B. After we finish the propagation in the second round, the nodes 11,12,13 will be labeled 1, and we can conclude that in this example, all calls can be served.

This example shows that if we initially assign k channels to k calls, then the propagation of 1's along the arcs is not enough to find the solution. In the example shown, after the first propagation phase is completed, nodes 8 and 10 are not labeled 1, but we noticed that they can actually be served. However, we may need less number of iterations than algorithm B. For instance, we need two rounds to find the solution set in this example but algorithm B will need three rounds because $k = 3$.

In order to find the nodes that are not labeled 1 during the propagation process but can be served, we can construct an array $M(1 \cdots n)$ such that $M(i)$ holds the number of incoming arcs to node i, which come from the nodes that have been labeled 1 during the propagation phase. For the example above, after finishing the first round of propagation we have the following array M. M contains 13 entries since node 14 is the last dummy node in the TPG.

1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	0	1	1	0	3	0	1	0	0	0	0

Now, for each call we want to compute the number of available channels except the one used by itself at its starting time. We thus subtract each entry by 1, except entries 1,3,6 that are labeled 1 at the beginning of the first round of propagation, and get the resulting array N

1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	2	-1	0	-1	-1	-1	-1

In this array, node 7 is the only node that has excess channels at its starting time. This fact is indicated by the positive value 2 in its corresponding entry in array N above.

Now, if we perform the prefix sums computation on the subrange $[7 \ldots 10]$, we will get the result

1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	2	1	1	0	-1	-1	-1

Observe that only entries 8 and 10 change their contents from negative values to nonnegative values. We may use this property to tell which calls should be set to 1 and become the starting nodes for the next round of propagation, even if they are not labeled 1 during the current round of propagation.

In general, the array N can be more complicated than the one above. For instance, for a problem with 19 calls and capacity 6, N may be the following array:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	4	0	0	0	-1	-1	-1	-1	-1	2	-1	-1	-1

In this array, we need to compute the prefix sums on the subranges $[7 \ldots 14]$ and $[16 \ldots 18]$. The result will be the following array.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	4	4	4	4	3	2	1	0	-1	2	1	0	-1

From this array, we deduce that nodes 11,12,13,14,17,18 should be assigned 1 at the start of the next round of propagation since the contents of their corresponding entries change from negative values to nonnegative values.

The above example contains two subranges. In general, the array N may contains many subranges. We formalize this computation by calling it Subrange Prefix Sums. We also use an array C to indicate which entry changes its contents from a negative value to a nonnegative value. We call array C the Characteristic Array of Subrange Prefix Sums.

Definition 4 Let $N(1 \cdot\cdot n)$ be an array, such that for each $N(i)$, $1 \leq i \leq n$, $N(i)$ is in the range $-1 \ldots k-1$, $k > 0$. Let $SB(1 \cdot\cdot n)$ be another array, such that $SB(i) = \sum_{j=l}^{i} N(j)$, $1 \leq i \leq n$, where $l \leq i$ and

(a). $N(l) > 0$.
(b). $\forall h, l \leq h < i, \sum_{j=l}^{h} N(j) > 0$ and,
(c). $\nexists\ l'$, $l' \leq l$ such that $N(l') \geq 1$ and l' satisfies (b)
(d). If no such l satisfies (a),(b),(c), then $l = i$.

Then, we call array SB the result of **subrange prefix sums (SPS)** on N. Also we call array SB the **SPS array** and call array N the **original array**.

Definition 5 Let SB be the result of subrange prefix sums on an array $N(1 \cdot\cdot n)$. Let $C(1 \cdot\cdot n)$ be another array such that for $1 \leq i \leq n$

$$C(i) = 1 \text{ if } SB(i) \geq 0 \text{ and } N(i) = -1;$$
$$C(i) = 0 \text{ otherwise.}$$

Then, we call array C the **Characteristic array of subrange prefix sums (CASPS)** on N.

We will give an efficient parallel algorithm to compute the SPS and the CASPS at the end of this section. For now, we assume we have an $O(\log n)$ time algorithm using $O(n/\log n)$ processors to find the CASPS on a given array N.

In what follows, we present a faster algorithm for the TCP which uses the algorithm for finding the CASPS as a subroutine.

Algorithm C A Faster algorithm for the TCP.

Input: TPG $G = (V, E)$, $\mid V \mid = n + 1$, and an integer $k < n$.

Output: Node i will be assigned 1 *iff* $i \in$ the solution set of G on k.

1. Assign label 1 to the leftmost k nodes that do not have an incoming arc. If the number of nodes that do not have an incoming arc is less than or equal to k, then assign label 1 to all nodes and stop.

Repeat the following steps:

2. Assign label 1 to all nodes that are reachable from any node that has received label 1.
3. $\forall i$, $1 \leq i \leq \mid V \mid -1$, $M(i)$:= number of incoming arcs to node i, which come from nodes that are currently labeled 1.
   $\forall i$, $1 \leq i \leq \mid V \mid -1$, if node i was labeled 1 in step 1 or step 8 then $N(i) := M(i)$ else $N(i) := M(i) - 1$.
4. Use algorithm D to construct the array C, the CASPS on N. (Algorithm D is given at the end of this section.)
5. $\forall i$, $1 \leq i \leq \mid V \mid -1$, if $C(i) = 1$ then assign label $1'$ to node i. ($1'$ is a temporary label.)
6. $\forall i$ not labeled 1, $1 \leq i \leq \mid V \mid -1$, if $(i, j) \in E$, nodes $j, j + 1, \ldots, j + m - 1$ have been labeled 1 or $1'$, and node $j + m$ has not been labeled 1 or $1'$, remove arc (i, j) and add arc $(i, j + m)$ to E.
7. Remove all nodes that have been labeled 1 and their outgoing arcs, and also remove all nodes that are not labeled 1 and are before the leftmost node with label $1'$ and their outgoing arcs to obtain a new TPG G.
8. Replace label $1'$ by label 1 for all nodes with label $1'$.

Until $\forall i$, $1 \leq i \leq \mid V \mid -1$, $C(i) = 0$.

End.

Definition 6 Let $S_i = \{i_1, i_2, \ldots, i_m\}$, where $i > 1$, be the set of nodes that were labeled 1 in step 8 during iteration $i-1$. We call S_i the **initial set** for iteration i. We let the set of nodes with label 1 at the end of step 1 be S_1, the initial set for iteration 1.

In what follows, we prove the correctness of algorithm C.

Lemma 4 *Let $S_1 = \{i_1, i_2, \ldots, i_k\}$ be the initial set for iteration 1, where k is the capacity of the telephone line. Then, $\forall i \leq i_k$, the corresponding call i can be served.*

Proof. Since $\forall i \leq i_k$, $i \notin S_1$, node i is reachable from a node $i_j \in S_1$, call i can use the channel that call i_j used. Hence, k channels are sufficient to serve all calls whose starting times are less than or equal to s_{i_k}. □

The next corollary follows from lemma 4.

Corollary 1 *If at the beginning of step 1, the number of nodes that do not have an incoming arc is less than or equal to k, then all calls can be served.*

Lemma 5 *A node is labeled 1 during the execution of algorithm C iff the corresponding call can be served.*

Proof. Let G_i denote the graph that algorithm C works on during iteration i. Let X_i denote the set of nodes that received label 1 before the start of iteration i. Also, let $k_i = \mid S_i \mid$ and let Y_i be the solution set of G_i with capacity k_i, where we interpret G_i as a TPG. We will prove the following claim by induction on the iteration number i.

Claim. For each iteration i, $X_i \cup Y_i$ is the solution set of the TPG G with capacity k, where G and k are inputs to algorithm C.

For the base case, since X_1 is empty and $G_1 = G$, the claim is trivially true. Assume the induction hypothesis is true until iteration $i-1$. Consider iteration i.

Let i'_1 and i_1 be the leftmost nodes in S_{i-1} and S_i, respectively. We first show that if a node j between i'_1 and i_1 in G is not labeled 1 at the end of iteration $i-1$ then call j cannot be served. Since j is not labeled 1 at the end of iteration $i-1$, j is not reachable from any node in S_{i-1} in G_{i-1}. Hence, we can find k_{i-1} arcs in G_{i-1} such that each one goes from a node before j to a node after j and both nodes are labeled 1 in step 2 during iteration $i-1$. This implies there is no available channels at time s_j, and j is not in the solution set Y_{i-1}. By the induction hypothesis, j is not in the solution set of G with capacity k.

Second, we show that X_i is contained in the solution set of G with capacity k. Since by the induction hypothesis X_{i-1} is contained in the solution set of G with capacity k, it suffices to show that all nodes in S_{i-1} and all nodes reachable in G_{i-1} from a node in S_{i-1} can be served, and all nodes in S_i can be served. By lemma 2, all nodes in S_{i-1} and all nodes reachable in G_{i-1} from a node in S_{i-1} are in Y_{i-1} and hence by the induction hypothesis they can be served.

Let N be the array computed in step 3 during iteration $i-1$. By the definition of SPS, for each $j \in S_i$, there exists the nearest entry $N(l)$ to $N(j)$, where $N(l) > 0$ and $l < j$, such that $\forall h, l \leq h < j, \sum_{g=h}^{j} N(g) > 0$. By the definition of array N, $N(l)$ is the number of available channels given to node l during iteration $i-1$ minus one that represents the channel used to serve l itself. By the definition of SPS,

one of the channels available for l is still available at time s_j and call j can be served. Hence, all nodes in S_i can be served.

Third, we show that a node in G_i is in Y_i iff it is in the solution set of G with capacity k. To show this, it suffices to show that the number of available channels for the nodes in G_i is k_i, and each arc (j, l) in G_i maintains the property that if j is served then l is served as stated in lemma 2.

From the construction of G_i, an arc is in G_i if the arc is in G_{i-1} or the arc is created in step 6 during iteration $i-1$. If an arc from j to l is in G_{i-1} then by the induction hypothesis this arc maintains the property that if j is served then l is served. If the arc is not in G_{i-1} then there exist nodes $h, h+1, \ldots, l-2, l-1$ in G_{i-1} such that the arc from j to h is in G_{i-1} and $h, h+1, \ldots, l-2, l-1$ are labeled 1 or $1'$ at the start of step 6 during iteration $i-1$. Note that $h, h+1, \ldots, l-2, l-1$ can be served since they are in X_i and we have shown that X_i is contained in the solution set of G with capacity k. From step 7 of algorithm C we know that j is not labeled 1 at the end of iteration $i-1$. Hence, if j can be served, then it must be the case that j uses a channel that none of $h, h+1, \ldots, l-2, l-1$ used. From this, we can conclude that if j can be served then l can use the channel that j uses and hence the arc from j to l in G_i maintains the property that if j is served then l is served.

We now show that the number of available channels for nodes in G_i at the start of iteration i is k_i. Since each node in S_i holds an available channel at the start of iteration i, it suffices to show that $\mid S_{i-1} \mid -k_i$ channels are unavailable at the start of iteration i. By the induction hypothesis, it suffices to consider only the channels available at the start of iteration $i-1$. We will prove the result by induction on k_{i-1}.

When $k_{i-1} = 1$, let $S_{i-1} = \{i'_1\}$. Hence, S_i is empty. The channel that i'_1 uses is always occupied by the calls that received label 1 in step 2 during iteration $i-1$. Thus, this channel is not available during iteration i.

Assume the induction hypothesis is true until $k_{i-1} = m-1$. Consider the case $k_{i-1} = m$. Let N'_{i-1} denote the SPS array that is obtained to compute the CASPS in step 4 during iteration $i-1$ if the initial set for iteration $i-1$ is the set $S'_{i-1} = \{i'_1, \ldots, i'_{m-1}\}$. Consider the case when the initial set for iteration $i-1$ is the set $S_{i-1} = \{i'_1, i'_2, \ldots, i'_m\}$. Let N_{i-1} denote the SPS array that is obtained to compute the CASPS in step 4 during iteration $i-1$ if the initial set for iteration $i-1$ is the set S_{i-1}. Let j be the first node that lies both on the path starting from i'_m and on a path starting from a node i'_l where $l < m$. This implies one of the following three cases:

case a. j is the last dummy node. The channel used by i'_m is always occupied by the calls on the path from i'_m to j. Hence this channel is not available at the start of iteration i. Thus, by the definition of SPS and by the induction hypothesis made with respect to k_{i-1}, k_i is the number of available channels for the remaining nodes in G_i at the start of iteration i.

case b. j is not the last dummy node and $\exists$ a nearest node q to j such that $j < q$ and $N'_{i-1}(q) = -1$. Then, by the definition of SPS, $\forall r, j \leq r \leq q, N_{i-1}(r) = N'_{i-1}(r) + 1$. Hence, the difference between the initial set for iteration i, which is obtained when the initial set for iteration $i-1$ is S_{i-1} and the initial set for iteration i, which is obtained when the initial set for iteration $i-1$ is S'_{i-1} is $\{q\}$. Since $\forall r, j \leq r < q, N'_{i-1}(r) \geq 0$, by the definition of SPS, q can use the channel that j uses. Since j can use the channel that i'_m uses, the channel used by i'_m is available during iteration i.

case c. j is not the last dummy node and $\nexists p$ such that $j < p$ and $N_{i-1}(p) = -1$. This implies all nodes after j received label 1 and were removed before the start of iteration i. Hence, the channel used by i'_m cannot be available for the calls processed during iteration i.

From the above argument, and the fact that the set of the first k_i nodes in G_i, which do not have an incoming arc is S_i, we can conclude that the solution set of G with capacity k is $X_i \cup Y_i$.

This completes the proof for the claim. Now let algorithm C stop after finishing iteration i, with inputs G and k. Since S_{i+1} is empty, Y_{i+1} is empty. Hence, by the claim X_i is the solution set of G with capacity k. □

From lemma 5 , we can conclude the following theorem.

Theorem 2 *Algorithm C correctly computes the solution set of an input TPG with capacity k.*

In what follows, we establish processor and time bounds for algorithm C.

Lemma 6 *Let S_i and S_{i+1} be the initial sets for iterations i and $i+1$, respectively. Then $\mid S_i \mid > \mid S_{i+1} \mid$.*

Proof. We prove the lemma by induction on the size of S_i. For $\mid S_i \mid = 1$, the result holds immediately. Assume the induction hypothesis holds until $\mid S_i \mid = l-1$ and consider $\mid S_i \mid = l$.

Consider a path from the leftmost node j in S_i. Let p be the first node that lies on this path and on the path starting from a node in S_i which is not j. If we remove the path from j to p but keep p, and run one iteration of algorithm C on the resulting graph, then by the induction hypothesis the initial set computed after that iteration will have size less than $l-1$.

Consider the case that we add back the path from j to p. Let N'_i be the SPS array computed first to obtain the CASPS result in step 4 during iteration i, without the path. Let N_i be the new SPS array when the path is added back. By the definition of SPS, we can find a nearest position q to the right of p such that $\forall r, p \leq r \leq q, N_i(r) = N'_i(r) + 1$. As a result, at most one more entry in N_i will change from negative values to nonnegative values. This implies the path from j to p will contribute at most one node to S_{i+1}. Thus, the size of S_{i+1} is less than l. □

Since algorithm C will terminate after finishing iteration i if S_{i+1} is empty, from lemma 6 we know that if $\mid S_i \mid = l$ then starting from iteration i, the Repeat loop of algorithm will execute no more than l times. We state this fact in the following corollary.

Corollary 2 *Starting from an iteration i, the Repeat loop of algorithm C will execute at most $\mid S_i \mid$ times.*

Lemma 7 *Algorithm C runs in time $O(k \log n)$ using n processors, assuming the CASPS can be computed in $O(\log n)$ time using $O(n/\log n)$ processors.*

Proof. From corollary 2 and from the fact that initially $\mid S_1 \mid \leq k$, we know that the Repeat loop of algorithm C executes at most k times. By the assumption that the algorithm of finding CASPS runs in time $O(\log n)$ using $O(n/\log n)$ processors, the only nontrivial step is step 2. Step 2 can be implemented using pointer jumping or using the *Euler Tour Technique* [8] in $O(\log n)$ time using n processors. □

Lemma 8 *Algorithm C runs in time $O(\sqrt{n}\log n)$, assuming the CASPS can be computed in $O(\log n)$ time using $O(n/\log n)$ processors.*

Proof. It suffices to show that the number of iterations of the Repeat loop of algorithm C is $O(\sqrt{n})$. Assume at iteration i, the initial set is S_i, $\mid S_i \mid = O(\sqrt{n})$, and for any $j < i$, S_j has size at least $\sqrt{n}$. Since $\mid S_1 \mid + \mid S_2 \mid + \ldots + \mid S_{i-1} \mid < n$, we have

$$(i-1)\sqrt{n} < n$$
$$\Rightarrow (i-1) < \sqrt{n}$$

By corollary 2, starting from iteration i, the **Repeat** loop will execute no more than $\mid S_i \mid$ times, so the total number of iterations is bounded by $O(\sqrt{n})$. □

We now give an optimal logarithmic time algorithm to compute the SPS and the CASPS. For this, we define a new associative binary operation $\star$ and convert the SPS problem into a prefix sums problem.

Definition 7 Given two pairs of integers $(-a, b), (-c, d)$, where $a, b, c, d \geq 0$, we define a binary operation $\star$, such that

$$\text{if } b - c > 0 \text{ then } (-a, b) \star (-c, d) = (-a, b - c + d)$$
$$\text{if } b - c \leq 0 \text{ then } (-a, b) \star (-c, d) = (-a + b - c, d)$$

Lemma 9 *The operation $\star$ is associative*

Proof. $((-a, b) \star (-c, d)) \star (-e, f)$, $a, b, c, d, e, f \geq 0$

$$= \begin{cases} b-c>0, b-c+d-e>0 & \Rightarrow (-a, b-c+d-e+f) \quad (1) \\ b-c>0, b-c+d-e\leq 0 & \Rightarrow (-a+b-c+d-e, f) \quad (2) \\ b-c\leq 0, d-e>0 & \Rightarrow (-a+b-c, d-e+f) \quad (3) \\ b-c\leq 0, d-e\leq 0 & \Rightarrow (-a+b-c+d-e, f) \quad (4) \end{cases}$$

and $(-a, b) \star ((-c, d) \star (-e, f))$

$$= \begin{cases} d-e>0, b-c>0 & \Rightarrow (-a, b-c+d-e+f) \quad (5) \\ d-e>0, b-c\leq 0 & \Rightarrow (-a+b-c, d-e+f) \quad (6) \\ d-e\leq 0, b-c+d-e>0 & \Rightarrow (-a, b-c+d-e+f) \quad (7) \\ d-e\leq 0, b-c+d-e\leq 0 & \Rightarrow (-a+b-c+d-e, f) \quad (8) \end{cases}$$

Note that the precondition part of (5) implies the precondition part of (1), and the precondition part of (7) also implies the precondition part of (1). Case (2) implies case (8) and case (4) implies case (8). Finally, case (3) and (6) are equivalent. By these facts, we can conclude that $\star$ is associative. □

Given an array $N(1 \cdot\cdot n)$ of integers in the range $[-1 \ldots k-1]$, we first construct a new array $M(1 \cdot\cdot n)$. For $1 \leq i \leq n$, $M(i)$ contains the pair $(0, b)$ if $N(i)$ contains b, $b \geq 0$, and $M(i)$ contains the pair $(-1, 0)$ if $N(i)$ contains -1. Then, we compute the prefix sums on M using the operation $\star$ defined above. The result is stored in another array $SB'(1 \cdot\cdot n)$ so for $1 \leq i \leq n$, $SB'(i) = \sum_{j=1}^{i} M(i)$ (where the summation uses the operation $\star$).

Observation 2 *For each i, $SB'(i) = (-a, b)$ with $a \geq 0, b \geq 0$.*

Lemma 10 *Let $SB'(1 \cdot\cdot n)$ be the array described above, and let $SB(1 \cdot\cdot n)$ be the result of subrange prefix sums on N. Then,*

$$SB(i) = \begin{cases} b & \text{if } SB'(i) = (-a, b) \text{ and } a \geq 0, b > 0 \\ 0 & \text{if } SB'(i) = (-a, 0) \text{ ,} i > 1 \text{ and } SB'(i-1) = (-c, 1), a, c \geq 0 \\ 0 & \text{if } SB'(i) = (-a, 0), a \geq 0 \text{ and } M(i) = (0, 0) \\ -1 & \text{otherwise} \end{cases}$$

Proof. We will prove the lemma by induction on i. For the base case we consider the following cases.

case a. $N(1) = b \geq 0$. Then $M(1) = (0, b)$ and $SB'(1) = (0, b)$. Hence, $SB(1) = b$.

case b. $N(1) = -1$. Then $M(1) = (-1, 0)$ and $SB'(1) = (-1, 0)$. Hence, $SB(1) = -1$.

Thus, we can conclude that

$$SB(1) = \begin{cases} b & \text{if } SB'(1) = (0, b) \text{ and } b > 0 \\ 0 & \text{if } SB'(1) = (0, 0), \text{ and } M(1) = (0, 0) \\ -1 & \text{otherwise} \end{cases}$$

Assume the result holds until $i-1$ and consider $SB(i)$.

case a. $M(i) = (0, b), b \geq 0$. By the induction hypothesis, we have one of the following three cases.

1. $SB'(i-1) = (-c, d)$, and $SB(i-1) = d > 0$. Then, $(-c, d) \star (0, b) = (-c, b+d)$ and $SB(i) = b+d$.
2. $SB'(i-1) = (-c, 0)$, and $SB(i-1) = 0$. Then, $(-c, 0) \star (0, b) = (-c, b)$ and $SB(i) = b$.
3. $SB'(i-1) = (-c, 0)$, and $SB(i-1) = -1$. Then, $(-c, 0) \star (0, b) = (-c, b)$ and $SB(i) = b$.

case b. $M(i) = (-1, 0)$. By the induction hypothesis, we have one of the following three cases.

1. $SB'(i-1) = (-c, d)$ and $SB(i-1) = d > 1$. Then, $(-c, d) \star (-1, 0) = (-c, d-1)$ and $SB(i) = d-1$.
2. $SB'(i-1) = (-c, d)$ and $SB(i-1) = 1$. Then, $(-c, 1) \star (-1, 0) = (-c, 0)$ and $SB(i) = 0$.
3. $SB'(i-1) = (-c, 0)$ and $SB(i-1) = d \leq 0$.
   Then, $(-c, 0) \star (-1, 0) = (-c-1, 0)$ and $SB(i) = -1$.

□

By lemma 10, we have the following algorithm to compute the CASPS.

Algorithm D Finding the characteristic array.

Input: Array $A(1 \cdots n)$ of elements from $\{-1, 0, 1, \ldots k-1\}$.

Output: Array $C(1 \cdots n)$, the CASPS of A.

1. Construct the array B from A as follow:
   If $A(i)$ contains b and $b \geq 0$, then $B(i) := (0, b)$.
   If $A(i)$ contains -1, then $B(i) := (-1, 0)$.
2. Compute the prefix sums on B using $\star$ as the binary operation, and store the result in $SB'(1 \cdots n)$.

3. For $1 \leq i \leq n$ in parallel do
   Let $SB'(i) = (-a, b)$ and $SB'(i-1) = (-c, d)$,$a, b, c, d \geq 0$.
   If $b > 0$ and $A(i) = -1$ then $C(i) := 1$
   else if $b = 0$ and $d = 1$ and $A(i) = -1$ then $C(i) := 1$
   else $C(i) := 0$.

End.

The following lemma is straightforward.

Lemma 11 *Algorithm D compute the CASPS on a given array $N(1 \cdot\cdot n)$ in $O(\log n)$ time using $O(n/\log n)$ processors on a EREW PRAM.*

Theorem 3 *Algorithm C runs in time $O(\min(k, \sqrt{n}) \log n)$ with n processors no matter what the capacity k is.*

Proof. By lemmas 7, 8, and 11. □

4 Reductions Between the TCP and the Comparator Circuit Value Problem

In this section, we present a log space reduction from the Comparator Circuit Value Problem (C-CV) to the TCP problem and a log space reduction from the TCP problem to the C-CV problem. Since the C-CV problem is CC-complete [7], we can conclude that the TCP problem is CC-Complete.

Definition 8 A comparator gate has two Boolean inputs u, v, and two outputs $uv, u + v$. Given a comparator circuit with m inputs, the C-CV problem is to decide the values of the circuit's m outputs.

The following algorithm converts an input to the C-CV problem to an input to the TCP problem.

Algorithm E Reduction from C-CV to TCP.

Input: A comparator circuit with $k+k'$ inputs and $k+k'$ outputs such that k is the number of ones in the inputs and k' is the number of zeros in the inputs. The inputs with value 1 are labeled $1, \ldots, k$ and the inputs with value 0 are labeled $1, \ldots, k'$. The gates are topologically ordered as $g_1, g_2, \ldots, g_n$.

Output: An instance of the TCP problem that has $k + k' + 2n$ calls, with capacity k.

1. Create calls $I = \{i_1, \ldots, i_k\}$ and calls $I' = \{i'_1, \ldots, i'_{k'}\}$ such that for each pair i_j, i'_l, the starting time of i_j is earlier than the starting time of i'_l. The relative order of the starting times among all calls in I is arbitrary. The relative order of the starting times among all calls in I' is also arbitrary.
2. $\forall$ gate g_i, create two calls i, i'. The starting times of these calls are later than the starting times of all calls in I and I'. Also let $s_1 < s_{1'} < s_2 < s_{2'} < \ldots < s_n < s_{n'}$.
3. $\forall$ pair of calls i, i' that correspond to gate g_i,

(a) if one of the inputs of g_i comes from the AND output of gate g_j, then let $f_{j'}$ (the finishing time of call j') occur immediately before s_i (the starting time of call i).

(b) if one of the inputs of g_i comes from the OR output of gate g_j, then put f_j immediately before s_i. The relative order between two finishing times put immediately before s_i can be arbitrary.

(c) if one of the inputs of g_i has value 1 with label j, then let the finishing time of call $i_j \in I$ occur immediately before s_i.

(d) if one of the inputs of g_i has value 0 with label l, then let the finishing time of call $i'_l \in I'$ occur immediately before s_i.

End.

We observe that for the instance of the TCP problem obtained from a comparator circuit by algorithm E, during any interval (s'_i, s_{i+1}), $1 \leq i \leq n-1$, there are two calls finishing that correspond to the two inputs to gate g_{i+1}, and during any interval $(s_i, s_{i'})$, there is no call finishing. Thus, we use two calls i, i' to simulate a gate g_i such that call i is used to simulate the OR output of g_i and call i' is used to simulate the AND output of g_i. Suppose we have a gate g_i whose inputs come from the AND output of gate g_j and the OR output of gate g_l. Then, we should have the resulting timing values in nondecreasing order as:

$$\ldots s_{i-1} s_{(i-1)'} f_l \; f_{j'} \; s_i \; s_{i'} \; \cdots$$

Here the relative order of f_l and $f_{j'}$ is unimportant. This example is illustrated in figure 3.

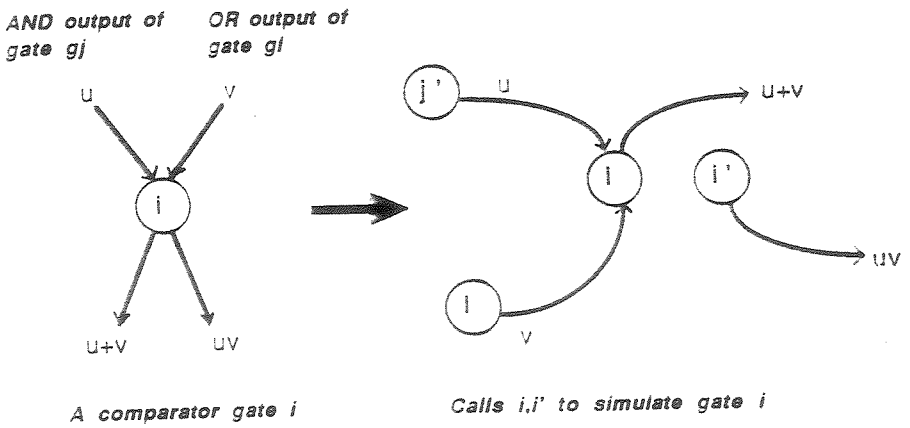

Figure 3: Transformation of a gate i to two calls i, i'.

Lemma 12 *Algorithm E converts an input to the C-CV problem to an input to the TCP problem such that for all i, call i is served iff the OR output of g_i has value 1, and call i' is served iff the AND output of g_i has value 1.*

Proof. By construction all calls in $I = \{i_1, \ldots, i_k\}$ can be served and all calls in $I' = \{i'_1, \ldots, i'_{k'}\}$ cannot be served.

We prove the lemma by induction on the gate number. For the base case, consider g_1.

If the OR output of g_1 has value 1, then one of its inputs has value 1. Hence there is an incoming arc to node 1, which comes from a node in I. Thus, call 1 can be served. If the AND output of g_1 has value 1, then both its inputs have value 1. Hence, there are two incoming arcs to node 1, which comes from nodes in I. Thus, at time s_1, there are two available channels released by two calls in I. This implies call $1'$ can be served.

Conversely, if call 1 can be served, then one of its incoming arcs should come from a node that can be served. This node is in I. Hence, one of the inputs to g_1 has value 1 and its OR output has value 1. If call $1'$ can be served, then both incoming arcs to node 1 should come from nodes that can be served. These nodes are in I. Hence, both inputs to g_1 have value 1 and its AND output has value 1.

Assume the induction hypothesis holds until $i-1$ and consider g_i.

If the OR output of g_i has value 1, then one of its inputs has value 1. Hence by the induction hypothesis there is an incoming arc to node i, which comes from a node that can be served. Thus, call i can be served. If the AND output of g_i has value 1, then both its inputs have value 1. Hence, by the induction hypothesis there are two incoming arcs to node i, which come from nodes that can be served. Thus, at time s_i, there are two available channels released by two calls that can be served. This implies call i' can be served.

Conversely, if call i can be served, then one of its incoming arcs should come from a node that can be served. Hence, by the induction hypothesis one of the inputs to g_i has value 1 and its OR output has value 1. If call i' can be served, then both incoming arcs to i should come from nodes that can be served. Hence, by the induction hypothesis both inputs to g_i have value 1 and its AND output has value 1. □

Algorithm E can be implemented in log space on a deterministic Turing machine. By lemma 12, we have the following theorem.

Theorem 4 *The C-CV problem is log space reducible to the TCP problem.*

From theorem 4, we conclude that the TCP problem is CC-hard. Next, we give a log space reduction from the C-CV problem to the TCP problem. This establishes the result that the TCP problem is CC-complete.

Let k be the capacity of the telephone line. From lemma 4, we know that any call before i_k, the kth node that does not have an incoming arc, can be served. Hence, if we change the starting time of i_k to be earlier than the starting times of the calls that have incoming arcs, this change will not affect the contents of the solution set. In fact, we may change the starting time of i_j, the jth node that does not have an incoming arc, to be earlier than the starting time of the calls that have incoming arcs, for each j, $1 \leq j \leq k$, and this change will not affect the contents of the solution set either since a call before i_j can be served using only $j-1$ channels. We state this property as the following observation.

Observation 3 *Let T_1 be an instance of the TCP with capacity k and let S_1 be the solution set for T_1. Let T_2 be obtained from T_1 by changing the starting times of the first k nodes that do not have an incoming arc in the TPG to be earlier than the starting times of any other calls and let S_2 be the solution set for T_2. Then, $S_1 = S_2$.*

From the above observation, we may assume that given an instance of the TCP with capacity k, the first k nodes do not have an incoming arc in the TPG. Next, we show how to reduce the TCP to C-CV based on this assumption. Figure 4 illustrates the construction in algorithm F.

Algorithm F Reduction from TCP to C-CV.

Input: An input to the TCP problem with capacity k such that the first k nodes do not have an incoming arc in the TPG. The calls are ordered from 1 to n.

Output: A comparator circuit with k inputs with value 1 and size $O(n^2)$.

1. Convert the given instance of the TCP problem to its TPG G. Assign label *served* to the k leftmost nodes that do not have an incoming arc.
2. Compute the number of incoming arcs to each node. Store the result in an array $N(1 \cdot \cdot n)$. For each node i, assign labels $1, \ldots, N(i)$ to its incoming arcs. This assignment can be done in arbitrary order. Also assign label *original* to the outgoing arc of node i.
3. $\forall i$, $1 \leq i \leq n$, $N''(i) := N(i) - 1$.
4. Compute subrange prefix sums on N'' using algorithm D. Store the result in another array N'. (For all $i > k$, $N'(i) + 1$ is the number of available channels at time s_i if all calls before i are served. We have $N'(i) = N'(i-1) + N(i) - 1$ when $N'(i-1) \geq 0$, and $N'(i) = N(i) - 1$ when $N'(i-1) < 0$. $N'(i)$ is in the range $-1 \ldots n-2$.)
5. $\forall i$, $k+1 \leq i \leq n$, let $N'(i) = M - 1$, $N'(i-1) = m'$, and $N(i) = m$. If $M > 1$, then create $M - 1$ arcs from node i to node $i+1$ and call the resulting graph the **modified graph**. (Note that $M = m + m'$ when $m' \geq 0$ and $M = m$ otherwise.)
6. For each i, $1 \leq i \leq n$, perform steps (a)-(e):
   (a) If $M > 1$, then create gates: $i_1, i_2, i_3, \ldots, i_{M-1}$.
      i. The right input to each gate i_j comes from the OR output of gate i_{j-1}, where $2 \leq j \leq M - 1$.
      ii. Let i' be a node in G whose outgoing arc with label *original* goes to node i and let l represent the number of outgoing arcs from node i' in the modified graph. Let $h = l - 1$ if $l \geq 2$ and let $h = 1$ otherwise. Let j be the label given to the arc from i' to i in step 2. If $j > 1$, let the left input of gate i_{j-1} come from the OR output of gate i'_h. If $j = 1$, let the right input of gate i_1 come from the OR output of gate i'_h.
      iii. The left input to gate i_{m+j-1} comes from the AND output of gate $(i-1)_j$, where $1 \leq j \leq m'$.

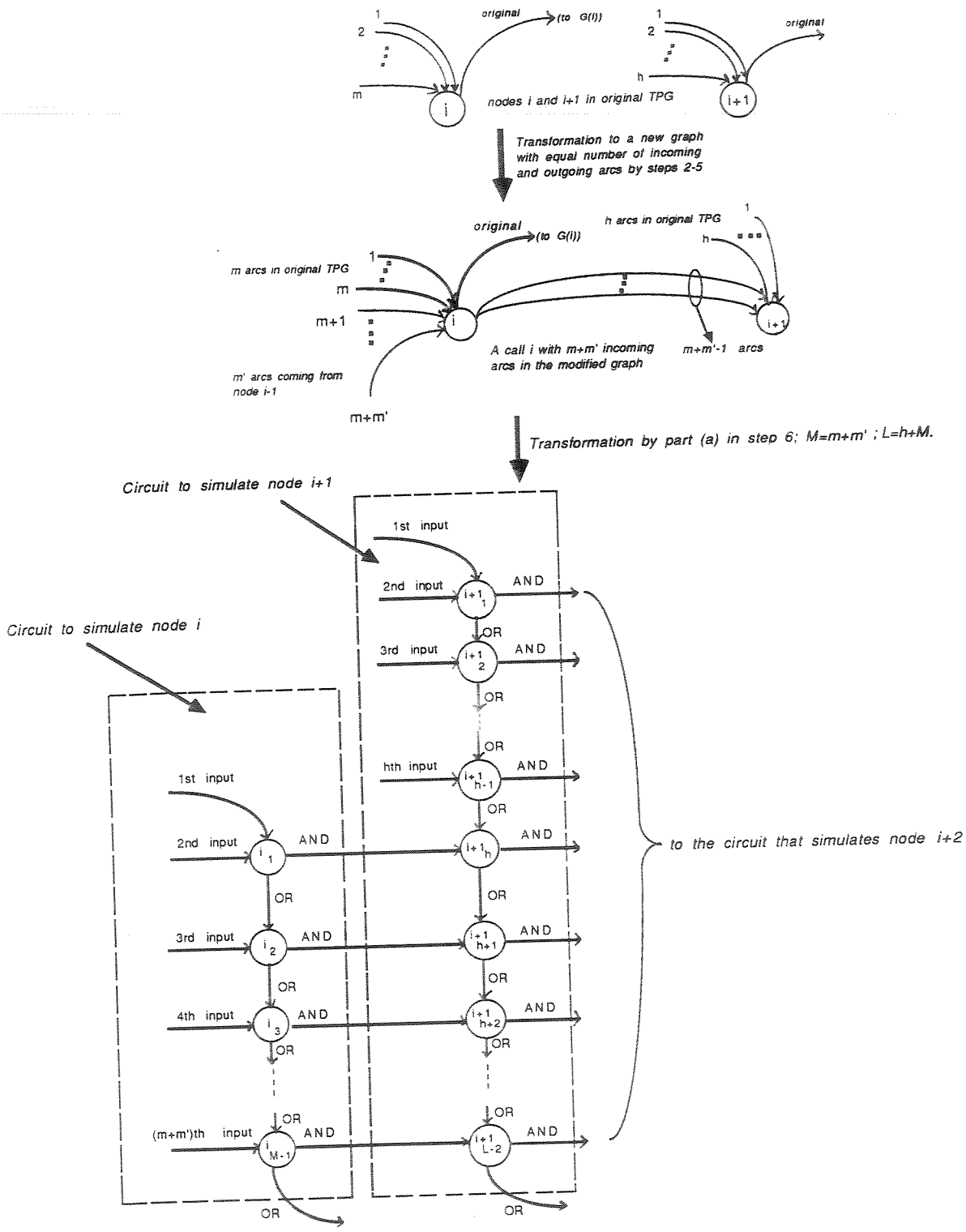

Figure 4: Illustration of the reduction in algorithm F.

(b) If $m = 1, m' = 0$, let the unique incoming arc to node i come from node i'. Create gate i_1. Let l represent the number of outgoing arcs from node i' in the modified graph. Let $h = l - 1$ if $l \geq 2$ and let $h = 1$ otherwise. The right input to gate i_1 has value zero. The left input to gate i_1 comes from the OR output of gate i'_h.

(c) If $m = 0, m' = 1$, then create gate i_1. The right input to gate i_1 has value zero. The left input to gate i_1 comes from the AND output of gate $(i-1)_1$, where node $i-1$ has two outgoing arcs in the modified graph.

(d) If $m + m' = 0$, and node i was labeled *served*, then create gate i_1. The left input to gate i_1 has value 1. The right input to gate i_1 has value 0.

(e) If $m + m' = 0$, and node i was not label *served*, then create gate i_1. Both inputs to gate i_1 have value 0.

End.

The above reduction from the TCP to C-CV could result in a quadratic blow-up in size since a call in an instance of the TCP may result in $\theta(n)$ comparator gates and an instance of the TCP may contain $\theta(n)$ such calls.

In what follows, we prove the correctness of algorithm F. We first make an observation that for any comparator gate circuit, the number of inputs with value 1 is equal to the number of outputs with value 1. Observe that each comparator gate has two inputs and two outputs and the number of inputs with value 1 to a gate is equal to the number of outputs with value 1 from that gate. A simple proof by induction on the number of gates in the circuit establishes the following observation.

Observation 4 *For any comparator gate circuit, the number of inputs with value 1 is equal to the number of outputs with value 1.*

Lemma 13 *Let the comparator circuit C with $m + m' = M, M > 1$, inputs be obtained by part (a) in step 6 of algorithm F for a node i. Then,*

(1). *The OR output of gate i_{M-1} has value 1 iff there is an input to circuit C with value 1.*

(2). *If none of the AND outputs from circuit C has value 1 then the number of inputs to circuit C with value 1 is either 1 or 0.*

(3). *For $j > 0$, exactly j AND outputs from gates $i_1, i_2, \ldots, i_{M-1}$ have value 1 iff there are exactly $j+1$ inputs to circuit C with value 1.*

Proof. Let an input to a gate i_j have value 1. Then, the OR outputs from gates $i_j, i_{j+1}, \ldots, i_{M-1}$ have value 1. Conversely, if the OR output of gate i_{M-1} has value 1, then we can always find a gate i_j such that one of its inputs has value 1, the OR outputs from gates $i_j, i_{j+1}, \ldots, i_{M-1}$ have value 1, and the OR outputs from gates $i_1, i_2, \ldots, i_{j-1}$ have value 0. Hence, (1) is true. It is clear that all inputs to C have value 0 iff all outputs from C have value 0. Then, (2) and (3) are true by (1) and observation 4. □

Lemma 14 *Algorithm F converts an input to the TCP problem to an input to the C-CV problem such that for all i, call i is served iff the OR output of gate i_L has value 1, where $L = M - 1$ if $M \geq 2$ and $L = 1$ otherwise, and M is as stated in step 5 of algorithm F.*

Proof. Let k be the capacity of the telephone line. From lemma 4, we know that the first k calls whose corresponding nodes in the TPG do not have an incoming arc can be served. Thus, for each gate i_1 created in part (d) of step 6, where $i \leq k$, its OR output has value 1 and call i can be served.

Let n be the number of calls. For the remaining calls $k+1, \ldots, n$, we prove the following claim by induction on the call number.

Claim The number of inputs with value 1 to the circuit that simulates a node i is the number of available channels at time s_i.

Before we prove the claim, first notice that a node without any incoming arc in the modified graph constructed in step 5 corresponds to a call that cannot be served and the circuit constructed in part (e) of step 6 to simulate this node has two zero inputs. Hence, the claim holds for the node. In the proof below, we eliminate this case and consider only the case when a node has at least one incoming arc in the modified graph constructed in step 5.

Consider the leftmost node that was not labeled *served* in step 1 of algorithm F. It should be the $(k+1)$th node by the assumption made about the input to algorithm F. Let C_{k+1} be the circuit that simulates node $k+1$. All inputs to C_{k+1} should have value 1. The OR output of gate j_1, where $1 \leq j \leq k$ and j_1 is created in part (d) of step 6 of algorithm F to simulate node j, is an input to C_{k+1} iff there is an arc from node j to node $k+1$ in the TPG. Hence, the number of inputs with value 1 to C is the number of available channels at time s_{k+1}.

Assume the induction hypothesis holds until call $i-1$. For call i, assume $N(i) = m, N'(i-1) = m'$ and assume C_i is the circuit with $M = m + m'$ inputs that simulates call i. In the proof below, let node j have M' outgoing arcs in the modified graph and let $L' = M' - 1$ if $M' \geq 2$ and let $L' = 1$ otherwise.

(1). Consider a fixed input to one of the gates $i_1, \ldots, i_{m-1}$ in C_i. Let this input come from the OR output of gate $j_{L'}$, where $1 \leq j < i$. By part (1) of lemma 13, this input has value 1 iff at least one of the inputs to gates $j_1, \ldots, j_{L'}$ has value 1, and by the induction hypothesis iff the number of available channels at time s_j is at least one, and hence iff call j can be served. Since call j can be served iff an available channel is released by call j in the interval (s_{i-1}, s_i), the number of inputs with value 1 to gates $i_1, \ldots, i_{m-1}$ is the number of available channels released in the interval (s_{i-1}, s_i).

(2). Suppose $m' > 0$. Consider the inputs to gates $i_m, i_{m+1}, \ldots, i_{m+m'-1}$ in C_i. For $p > 0$, from part (3) of lemma 13, the number of inputs with value 1 to gates $i_m, i_{m+1}, \ldots, i_{m+m'-1}$ in C_i is p iff the number of inputs with value 1 to the circuit that simulates $i-1$ is $p+1$, and by the induction hypothesis, iff the number of available channels at time s_{i-1} is $p+1$. Among these channels, p channels are still available at time s_i.

From part (2) of lemma 13, all inputs to gates $i_m, i_{m+1}, \ldots, i_{m+m'-1}$ in C_i have value 0 iff the number of inputs with value 1 to the circuit that simulates $i-1$ is either 1 or 0, and by the induction hypothesis, iff the number of available channels at time s_{i-1} is either 1 or 0. In this case, no available channel at time s_{i-1} is still available at time s_i. Hence, the number of inputs with value 1 to gates $i_m, \ldots, i_{m+m'-1}$ is the number of available channels at time s_i, which are released before time s_{i-1}.

By (1) and (2), the number of inputs with value 1 to C_i is equal to the number of available channels at time s_i. This establishes the induction step and the claim is proved. Then, by the definition of the TCP,

the above claim, and part (1) of lemma 13, we conclude that call i can be served iff there is an available channel at time s_i, iff there is an input with value 1 to C_i, and iff the OR output of gate i_{M-1} has value 1. Hence, the lemma holds for calls $k+1, \ldots, n$. □

Algorithm F can be implemented in log space on a deterministic Turing machine. Thus, by lemma 14, we have the following theorem.

Theorem 5 *TCP is log space reducible to the C-CV problem.*

Corollary 3 *TCP is CC-complete.*

Proof. by theorem 4 and theorem 5. □

From algorithm E and lemma 12, we know that every input of size n to the C-CV problem can be converted to an instance of the TCP problem of size $O(n)$, and thus to a TPG such that each node has at most 2 incoming arcs. Also, for an input to the C-CV problem, if the number of inputs with value 1 is k, then algorithm C can be used to solve the C-CV problem in time $O(\min(k, \sqrt{n}) \log n)$ using n processors. By the symmetric property of the comparator gate, if the number of inputs with value 0 is k', then algorithm C can also be used to solve the C-CV problem in time $O(\min(k', \sqrt{n}) \log n)$ using n processors. Hence, algorithms C and E can solve the C-CV problem in time $O(\min(k, k', \sqrt{n}) \log n)$ using n processors. The previous best algorithm for this problem [7] runs in $O(\sqrt{n} polylog)$ time.

References

[1] R. Cole. *Parallel merge sort*, SIAM J. on Computing, Vol 17, No. 4, August 1988, pp 770-785.

[2] A. Greenberg. *Private communication*, June 1990.

[3] A. Greenberg. *Private communication*, August 1990.

[4] D. S. Johnson. *A catalog of complexity classes*, Handbook of Theoretical Computer Science, Vol 1, Elsevier Press, 1990, pp 67-161.

[5] R. Karp and V. Ramachandran. *Parallel algorithms for shared-memory machine*, Handbook of Theoretical Computer Science, Vol 1, Elsevier Press, 1990, pp 869-941.

[6] R. Ladner and M. Fischer. *Parallel prefix sum*, JACM, Vol 27, 1980, pp 831-838.

[7] E. Mayr and A. Subramanian. *The complexity of circuit value and network stability*, Proc. 4th Annual Conf. on Structure in Complexity Theory, 1989, pp 114-123.

[8] R. Tarjan and U. Vishkin. *Finding biconnected components and computing tree functions in logarithmic parallel time*, SIAM J. on Computing, Vol 14, 1985, pp 862-874.