REALIZING DEGREE SEQUENCES IN PARALLEL*

SRINIVASA R. ARIKATI† AND ANIL MAHESHWARI‡

Abstract. A sequence d of integers is a degree sequence if there exists a (simple) graph G such that the components of d are equal to the degrees of the vertices of G. The graph G is said to be a realization of d. We provide an efficient parallel algorithm to realize d; the algorithm runs in $O(\log n)$ time using $O(n+m)$ CRCW PRAM processors, where n and m are the number of vertices and edges in G. Before our result, it was not known if the problem of realizing d is in NC.

Key words. design and analysis of algorithms, parallel computation, graph algorithms, degree sequence, majorization, PRAM

AMS subject classifications. 68Q22, 68R10

1. Introduction.

1.1. Problem Definition. An important problem in graph algorithms is to compute a (simple undirected) graph satisfying the given degree constraints. An integer sequence d of length n is called a *degree sequence* if there exists a graph G on n vertices such that the degrees of its vertices are equal to the components of the sequence d. The graph G is said to be a *realization* of the sequence d. A pair (r, s) of integer sequences is called a *bipartite sequence* if there exists a bipartite graph $H = (X \cup Y, E)$ such that the components of r (respectively, s) are equal to the degrees of the vertices in X (respectively, Y). Degree sequences and bipartite sequences have been extensively studied in graph theory [6, 15, 21, 26]. Because of the strong connections between the structural properties of a graph and the degrees of its vertices, these sequences find significant applications in the areas of communication networks, structural reliability, and stereochemistry (cf. [7, 26]).

1.2. Previous Results. Given an integer sequence d, there are two problems of interest: the *decision problem* is to test if d is realizable; the *search problem* is to compute a realization of d. A characterization of degree sequences known as the Erdős-Gallai inequalities [10] results in an efficient sequential algorithm for the decision problem. Another characterization called the Havel-Hakimi characterization (cf. [15]) leads to an efficient sequential algorithm for the search problem. In the case of bipartite sequences, a characterization known as the Gale-Ryser theorem [13, 22, 24] leads to efficient sequential algorithms for the decision as well as the search problems. Recently, degree sequence problems have gained lot of attention, see for example [2, 3, 4, 9, 21, 23, 25, 26].

The Erdős-Gallai inequalities and the Gale-Ryser theorem imply efficient parallel algorithms for the decision problems on degree sequences and bipartite sequences, respectively. Recently, a parallel algorithm for a special case of the search problem, in which the maximum degree is bounded by the square-root of the sum of the degrees, is presented in [9]; it runs in $O(\log^4 n)$ time using $O(n^{10})$ EREW PRAM processors.

* This work is partially supported by the EEC ESPRIT Basic Research Action No. 7141 (ALCOM II). A part of this paper appeared in *5th International Symposium on Algorithms and Computation, China, 1994*.

† MPI Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (arikati@mpi-sb.mpg.de).

‡ MPI Informatik, Germany. Author's current address: School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada (maheshwa@chaos.scs.carleton.ca).

Network-flow based proofs [11, 12] give rise to randomized parallel algorithms for the search problems on degree sequences and bipartite sequences.

1.3. Our Results. The main contributions of this paper are deterministic parallel algorithms for the search problems on degree sequences and bipartite sequences. Our results are:

- An efficient parallel algorithm for realizing bipartite sequences; it runs in $O(\log n)$ time using $O(n)$ EREW PRAM processors, where n is the number of vertices in the realization.
- A new proof of a relation between degree sequences and bipartite sequences.
- An efficient parallel algorithm for realizing degree sequences; it runs in $O(\log n)$ time using $O(n+m)$ CRCW PRAM processors, where n and m denote the number of vertices and edges in the realization.

The complexity results of this paper are with respect to the PRAM computational model. For details on the PRAM and NC, see [18, 19]. The *work*, i.e. *time*$\times$*processor* product, of our parallel algorithm for realizing bipartite sequences is $o(n^2)$, whereas there are bipartite graphs that have $\Omega(n^2)$ edges (e.g. a complete bipartite graph on n vertices). The complexity results of this paper are feasible since the graphs computed by our algorithms possess implicit representations, i.e., the graphs can be stored in $O(n)$ space, and the adjacency information between any two vertices can be reported in constant time [27].

Our result for realizing bipartite sequences is based on a non-trivial parallelization of the techniques from the theory of majorization [16, 22]. Our algorithm for realizing a degree sequence d is based on a new proof of a relation between degree sequences and bipartite sequences and it proceeds as follows. From d, we compute an appropriate bipartite sequence (c, c), and then compute a realization H of (c, c). Using the graph H, we compute a symmetric bipartite graph that leads to a realization of d. The computation of the symmetric bipartite graph from H is the crucial step, for which we provide two alternate parallel algorithms: the first one has higher complexity than the second. The latter algorithm exploits the implicit structure of the bipartite graph H computed by our algorithm and thus is efficient. The former algorithm does not assume any structural knowledge of H, and can work with any realization of (c, c). It is based on several interesting lemmas, which may be of independent interest in their own right.

1.4. Organization of the Paper. The rest of the paper is organized as follows. In Section 2 we introduce notation and state preliminaries. In Section 3 we state some of the classical characterizations of degree sequences and present simple algorithms for realizing the degree sequences corresponding to multigraphs and trees. In Section 4 we prove a relation between degree sequences and bipartite sequences. In Section 5 we present the required results from the theory of majorization, including an algorithm for computing unit transformations. In Section 6 we present a parallel algorithm for realizing bipartite sequences. In Section 7 we provide parallel algorithms for realizing degree sequences.

2. Preliminaries.

2.1. Basic Definitions. In a *multigraph* $G = (V, E)$, V is a set of vertices and E is a multiset of edges (multiple edges may exist between two vertices but no self-loops). By a *graph* $G = (V, E)$, we mean a simple graph—without multiple edges and self-loops. A bipartite graph H with the bipartition $X \cup Y$ is denoted by $H = (X \cup Y, E)$. In a multigraph $G = (V, E)$, $d_G(v)$ denotes the degree of a vertex v and $N_G(v)$

denotes the multiset of the neighbors of v (we omit the subscript, if no confusion arises). Similarly, $N_G(U)$ is defined as the union of the neighbors of the vertices of $U \subseteq V$. By definition, G is a graph if and only if the following hold for all $v \in V$: (i) $v \notin N_G(v)$ and (ii) $N_G(v)$ is a set. If (u, v) is an edge of a graph G, we say that $(u, v) \in G$. Throughout, by a sequence we mean a sequence of nonnegative integers.

2.2. Graph Matching. A *matching* M in a graph $G = (V, E)$ is a collection of edges such that no two edges of M are incident at a common vertex. The size of M, denoted by $|M|$, is the number of edges in it. M is called a *perfect matching* if it matches all vertices of G. M is called a *maximum matching* if it has maximum size among all matchings. M is called a *maximal matching* if no other matching properly contains M. We will need the following theorem due to P. Hall (cf. [21]).

THEOREM 2.1 (THE HALL'S THEOREM). *Let $H = (X \cup Y, E)$ be a bipartite graph. Then H has a matching that matches all vertices of X, if and only if $|N(A)| \geq |A|$ for every $A \subseteq X$.*

We need the following two lemmas; the first lemma can be proved easily.

LEMMA 2.2. *Let M (respectively, $M^{'}$) be a maximal (respectively, maximum) matching in a graph G. Then $|M| \geq \frac{1}{2}|M^{'}|$.*

LEMMA 2.3. *Let $H = (X \cup Y, E)$ be a bipartite graph such that (i) $d(x) \geq 1$ for all $x \in X$ and (ii) the inequality $d(x) \geq d(y)$ holds for every edge (x, y) of H. Then H has a matching that matches all vertices of X.*

Proof. We show that H satisfies the sufficiency part of the Hall's theorem. We use induction on $|A|$, where $A \subseteq X$. Since $d(x) \geq 1$ for all $x \in X$, the basis case, $|A| = 1$, follows. Consider now the case that $|A| = k$, where $k \geq 2$. We need to prove that $|N(A)| \geq k$. Assume the contrary, namely that $|N(A)| < k$. Pick any vertex $z \in A$ and put $A^{'} = A - z$. By induction, $|N(A^{'})| \geq k - 1$. Since $N(A^{'}) \subseteq N(A)$, it follows $(A^{'}) = N(A)$ and $|N(A^{'})| = |N(A)| = k - 1$. Consider now the subgraph $H^{'} = (A^{'} \cup N(A^{'}), E^{'})$ of H. By induction, $H^{'}$ satisfies the sufficiency condition of the Hall's theorem and hence has a perfect matching. Let the edges of the perfect matching be $(x_1, y_1), (x_2, y_2), \ldots, (x_{k-1}, y_{k-1})$. Using condition (ii) of the lemma, we obtain $\sum_{x \in A^{'}} d(x) = \sum_{i=1}^{k-1} d(x_i) \geq \sum_{i=1}^{k-1} d(y_i) = \sum_{y \in N(A^{'})} d(y)$. Since $d(z) \geq 1$ and $N(A) = N(A^{'})$, we have $\sum_{x \in A} d(x) > \sum_{y \in N(A)} d(y)$, which contradicts $\sum_{x \in A} d(x) \leq \sum_{y \in N(A)} d(y)$; the latter inequality holds for any bipartite graph because every edge incident to a vertex in A contributes a 1 to both sides of the inequality. This completes the induction step and hence the lemma. □

2.3. Digraphs. In a digraph $D = (V, E)$, E is the set of arcs (directed edges); the arc from u to v will be denoted by the ordered pair (u, v). The indegree (respectively, outdegree) of a vertex v, denoted by $d_D^-(v)$ (respectively, $d_D^+(v)$), is the number of arcs into (respectively, from) v. Call D *symmetric* if it has only symmetric arcs: (u, v) is an arc if and only if (v, u) is an arc. We will require the following lemma.

LEMMA 2.4. *([12]) Let $\tilde{D} = (V, \tilde{E})$ be a digraph such that indegree of each vertex v equals its outdegree, i.e., $d_{\tilde{D}}^-(v) = d_{\tilde{D}}^+(v) = d(v)$, and that $\sum_{v \in V} d_{\tilde{D}}^+(v)$ is even. Then there exists a symmetric digraph $D = (V, E)$ such that $d_D^-(v) = d_D^+(v) = d(v)$.*

Proof. If $\tilde{D}$ is symmetric, then take $D := \tilde{D}$. Assume that $\tilde{D}$ is not symmetric and let $\hat{D} = (V, \hat{E})$ be the 'asymmetric part' of $\tilde{D}$: $(u, v) \in \hat{E}$ iff $(u, v) \in \tilde{E}$ and $(v, u) \notin \tilde{E}$. Observe that $d_{\hat{D}}^-(v) = d_{\hat{D}}^+(v)$ for all $v \in V$. Define a *trail* to be a sequence of (not necessarily distinct) vertices $v_1, \ldots, v_k, v_1$ such that $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)$ are distinct arcs of $\hat{D}$. Call a trail *even* if it consists of an even number of edges, otherwise it is an *odd* trail.

Assume that $\hat{D}$ has an even trail, say $v_1, v_2, \ldots, v_{2k}, v_1$. Change $\tilde{D}$ as follows: Delete the arcs (v_2, v_3), (v_4, v_5), ..., (v_{2k-2}, v_{2k-1}), (v_{2k}, v_1) and add the arcs (v_2, v_1), (v_4, v_3), ..., (v_{2k-2}, v_{2k-3}), (v_{2k}, v_{2k-1}). Notice that this process does not create any multiple arcs or self-loops in $\tilde{D}$. Further, the hypothesis of the lemma is maintained. We repeat the above-mentioned process until $\hat{D}$ contains no even trails. Then, it follows that $\sum_{v \in V} d^+_{\hat{D}}(v)$ is even as $\sum_{v \in V} d^+_{\tilde{D}}(v)$ is even. Further, using $d^-_{\hat{D}}(v) = d^+_{\hat{D}}(v)$ for $v \in V$, we can decompose the arcs of $\hat{D}$ into odd directed cycles. There are an even number of such cycles. Moreover, any two such cycles must be vertex-disjoint, otherwise they create an even trail. Let $u_0, u_1, \ldots, u_{2k}, u_0$ and $v_0, v_1, \ldots, v_{2\ell}, v_0$ be any two odd cycles in $\hat{G}$. The fact that $(u_0, v_0) \notin \hat{D}$ implies that either both (u_0, v_0) and (v_0, u_0) are arcs in $\tilde{D}$ or none is an arc in $\tilde{D}$. We distinguish between these two cases.

Case 1: Both (u_0, v_0) and (v_0, u_0) are arcs in $\tilde{D}$. Change $\tilde{D}$ as follows: Delete the arcs (u_0, v_0), (v_0, u_0), (u_1, u_2), (u_3, u_4), ..., (u_{2k-1}, u_{2k}) and (v_1, v_2), (v_3, v_4), ..., $(v_{2\ell-1}, v_{2\ell})$; add the arcs (u_2, u_1), (u_4, u_3), ..., (u_{2k}, u_{2k-1}) and (v_2, v_1), (v_4, v_3), ..., $(v_{2\ell}, v_{2\ell-1})$.

Case 2: None of (u_0, v_0) and (v_0, u_0) is an arc in $\tilde{D}$. Change $\tilde{D}$ as follows: Delete the arcs (u_0, u_1), (u_2, u_3), ..., (u_{2k}, u_0), and (v_0, v_1), (v_2, v_3), ..., $(v_{2\ell}, v_0)$; add the arcs (u_0, v_0), (v_0, u_0), (u_2, u_1), (u_4, u_3), ..., (u_{2k}, u_{2k-1}) and (v_2, v_1), (v_4, v_3), ..., $(v_{2\ell}, v_{2\ell-1})$.

In each case no multiple arcs or self-loops are created, and the number of odd cycles in $\tilde{D}$ decreases. Eventually, $\tilde{D}$ contains no odd cycles and hence it becomes symmetric. The proof is completed by taking $D := \tilde{D}$. □

2.4. Parallel Techniques. The complexity results of this paper are with respect to the PRAM. This is the synchronous parallel model in which all processors share a common memory. In this paper, we need the following techniques previously developed in parallel computing: Euler tour in a graph [5], merging and cross-ranking [14], sorting [8], maximal matching in a graph [17]. For other techniques such as parallel prefix and list ranking, see [18, 19].

Consider a sequence of n elements $\{x_1, x_2, ..., x_n\}$ drawn from a set S with a binary associative operation $*$. The *prefix sums* of this sequence are the n partial sums (or products) defined by $s_i = x_1 * x_2 * ...x_i$, $1 \leq i \leq n$. Consider a linked list L of n nodes whose order is specified by an array S such that $S(i)$ contains a pointer to the node following node i on L, for $1 \leq i \leq n$. We assume $S(i) = 0$ when i is the end of the list. The *list ranking* problem is to determine the distance of each node i from the end of the list. The *rank* of an element x in a given sequence X is the number of elements of X that are less than or equal to x. Let A and B be two sorted sequences. The *cross-ranking* problem is to find the rank of each element of A in B and vice-versa.

3. Characterizations and Algorithmic Aspects.

3.1. Multigraphs. Realizability problems, in general, tend to be simpler if multiple edges are allowed. We show that this is the case in parallel computation too. We first discuss realizing degree sequences of bipartite multigraphs and then show how to reduce the general case to the bipartite case. Recall that in a multigraph, multiple edges may exist between a pair of vertices but self-loops are not allowed.

Let (r, s), where $r = (r_1, \ldots, r_m)$ and $s = (s_1, \ldots, s_n)$, be a pair of sequences. Our problem is to compute a bipartite multigraph $H = (X \cup Y, E)$ satisfying the degree constraints r and s. It is easy to prove that H exists if and only if $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$.

To realize (r, s) in parallel, test if $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$ and stop if the test fails. Then, compute the prefix sums of r and s and store them in the arrays R and S, respectively. Cross-rank S in R using the algorithm of [14]. Connect y_j to all the corresponding x_i's using the required number of multiple edges.

We now discuss the general case. Given a sequence d, the problem is to compute a multigraph with degree sequence d. The following lemma characterizes d [6, 15, 21]. The proof given below results in a simple parallel algorithm.

LEMMA 3.1. *The sequence $d = (d_1, \ldots, d_n)$, where $d_1 = \max(d)$, is the degree sequence of a multigraph if and only if $\sum_{i=1}^{n} d_i$ is even and $d_1 \leq \sum_{i=2}^{n} d_i$.*

Proof. We prove the sufficiency part, the other part being trivial. Sort the sequence d into nonincreasing order, i.e., let $d_1 \geq d_2 \geq \cdots \geq d_n$. Let $v_1, v_2, \ldots, v_n$ be the vertices of the multigraph G to be computed. Put $m = \frac{1}{2}\sum_{i=1}^{n} d_i$ and let p be the index such that $\sum_{i=1}^{p} d_i \leq m < \sum_{i=p+1}^{n} d_i$. We distinguish between two cases.

Case 1: $\sum_{i=1}^{p} d_i = m$. Define sequences r and s by $r = (d_1, d_2, \ldots, d_p)$ and $s = (d_{p+1}, \ldots, d_n)$. Then r and s have the same component sum and thus (r, s) can be realized, using the procedure given above, as a bipartite multigraph $G = (X \cup Y, E)$, where $X = \{v_1, \ldots, v_p\}$ and $Y = \{v_{p+1}, \ldots, v_n\}$.

Case 2: $\sum_{i=1}^{p} d_i < m$. Put $k = \sum_{i=p+1}^{n} d_i - m$ and define sequences r and s as follows: $r = (d_1 - k, d_2, \ldots, d_p, d_{p+1} - k)$ and $s = (d_{p+2}, \ldots, d_n)$. Then r and s have the same component sum $(= m-k)$ and thus (r, s) can be realized, using the procedure given above, as a bipartite multigraph $H = (X \cup Y, E)$, where $X = \{v_1, \ldots, v_{p+1}\}$ and $Y = \{v_{p+2}, \ldots, v_n\}$. By adding k multiple edges between v_1 and v_{p+1} in H we get the required multigraph G. □

3.2. Trees. We now discuss the degree sequences of trees. The following characterization of such sequences is well known [6, 15, 21]. We will present a proof that leads to a simple parallel algorithm to realize these sequences.

LEMMA 3.2. *The sequence $d = (d_1, \ldots, d_n)$ is the degree sequence of a tree if and only if all d_i's are positive and $\sum_{i=1}^{n} d_i = 2n - 2$.*

Proof. The necessity part is trivial and we prove the other part. Sort the sequence d into nonincreasing order; denote the resulting sequence also by d. Let $v_1, v_2, \ldots, v_n$ be the vertices of the tree G to be computed. If $d_1 = 2$ then G is a path and we are done. So assume that $d_1 \geq 3$ and let k be the largest index such that $d_k \geq 3$. Put $m = \sum_{i=1}^{k}(d_i - 2)$. Let A be the set of d_i's that are equal to 1. The following claim will be proved after we describe the computation of G.

Claim: $|A| = m + 2$.

First we compute a path consisting of vertices $v_{n-1}, v_1, v_2, \ldots, v_{n-m-2}, v_n$[1]. Delete v_n and v_{n-1} from A. Then we compute 'stars' using A and $v_1, v_2, \ldots, v_k$ as follows: Connect the first $d_1 - 2$ vertices of A to v_1, connect the next $d_2 - 2$ vertices of A to v_2, ..., and connect the remaining $d_k - 2$ vertices of A to v_k. This completes the computation of G.

We now prove the above claim. Observe that $\sum_{i=1}^{k} d_i = m+2k$ and that there are $n-k-|A|$ d_i's that are equal to 2. So $2n-2 = \sum_{i=1}^{n} d_i = m+2k+2(n-k-|A|)+|A|$. The claim follows by rearranging the terms. □

3.3. Graphs. We now discuss degree sequences of graphs. Let d be an integer sequence of length n, where $n > d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. The proofs of the following results may be found, e.g., in [6, 15, 21].

[1] Observe that $d_{n-1} = d_n = 1$.

The Erdős-Gallai Inequalities (EGI): The sequence d is realizable if and only if $\sum_{i=1}^{n} d_i$ is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ for $k = 1, 2, \ldots, n$.

The Havel-Hakimi Characterization: The sequence d is realizable if and only if the numbers $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n$ are realizable.

Using EGI, we can test if d is realizable in linear time. We can use the second characterization to derive an efficient sequential algorithm to compute a realization of d.

In parallel computation, the inequalities of EGI can be tested optimally, and this implies an optimal parallel algorithm to test if d is a degree sequence. As for the problem of computing a realization of d, a proof of the EGI using network flows is given in [12] and this proof results in a randomized parallel algorithm. The proof consists of two steps.
Step 1: Define a network with edge capacities in unary and solve the maximum flow problem on this network; then, construct a digraph D.
Step 2: Obtain a symmetric digraph from D.
Results of [20] imply a randomized parallel algorithm for Step 1. Based on the proof of Lemma 2.4, Step 2 can be performed in NC.

4. Degree Sequences and Bipartite Sequences. We study in this section a relation between degree sequences and bipartite sequences. The main result of this section is a new proof of a theorem given in [25]. The proof presented in [25] is via a cycle of eight implications and results in an inherently sequential algorithm to compute realizations of degree sequences, whereas our proof is simple and helps us to design a parallel algorithm.

Throughout this section, d denotes an integer sequence of length n, where $n > d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. Let $\mu = \max\{k : d_k \geq k\}$. Define a new sequence $c = (c_1, \ldots, c_n)$, where

$$c_i = \begin{cases} d_i + 1 & \text{if } i \leq \mu \\ d_i & \text{otherwise.} \end{cases}$$

Observe that $c_i = d_i + 1 \geq \mu + 1$ for $1 \leq i \leq \mu$ and $c_j = d_j \leq \mu$ for $\mu + 1 \leq j \leq n$.

THEOREM 4.1. *The sequence d is a degree sequence if and only if (c, c) is a bipartite sequence.*

Before we prove this theorem, we remark that the '+1' is required in the definition of c, since there are sequences d such that (d, d) is a bipartite sequence but d is not a degree sequence; for example, take $d = (4, 2, 2, 2)$. The proof is based on the following lemmas.

LEMMA 4.2. *If d is a degree sequence then (c, c) is a bipartite degree sequence.*

Proof. Let $G = (V, E)$ be a realization of d. We obtain a bipartite realization $H = (X \cup Y, E')$ of (c, c) as follows: X and Y are two copies of V; if $(v_i, v_j) \in G$ then $(x_i, y_j), (x_j, y_i) \in H$; further, $(x_i, y_i) \in H$ for all $1 \leq i \leq \mu$. □

We need a few definitions before presenting the next lemmas. Let $H = (X \cup Y, E)$ be a realization of (c, c). A vertex x_i (or y_i) is called a *high-degree* vertex if $1 \leq i \leq \mu$, otherwise it is a *low-degree* vertex. An edge (x_i, y_i) is called a *high-degree* edge if $1 \leq i \leq \mu$. Similarly, an edge (x_j, y_j) is called a *low-degree* edge if $\mu + 1 \leq j \leq n$. Edges of the form (x_i, y_j), where $i \neq j$, are neither high-degree edges nor low-degree edges. High-degree and low-degree edges play a very important role in our algorithms.

A pair of edges (x_α, y_β) and (x_γ, y_δ) form an *exchange pair* if $(x_\alpha, y_\delta), (x_\gamma, y_\beta) \notin H$ (see Figure 1). An *exchange* on the edges (x_α, y_β) and (x_γ, y_δ) consists of deleting

(x_α, y_β) and (x_γ, y_δ) and inserting (x_α, y_δ) and (x_γ, y_β). The following two lemmas imply that, given any realization H of (c, c), we can always obtain another realization $H^{'}$ such that $H^{'}$ contains all high-degree edges and no low-degree edges.

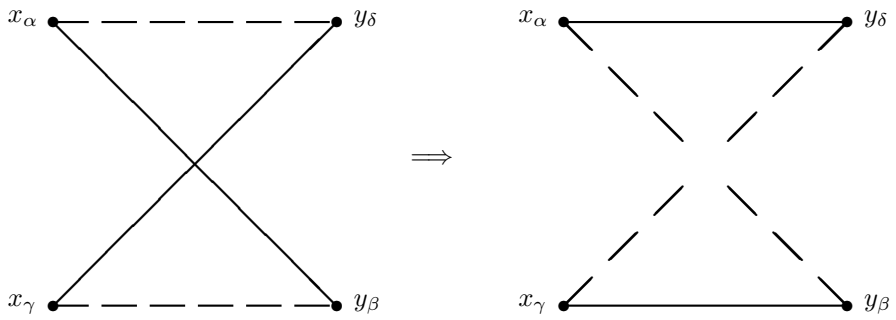

FIG. 1. *An exchange operation. A solid line indicates the presence of an edge and a dashed line its absence.*

LEMMA 4.3. *Let H be a realization of (c, c) such that $(x_i, y_i) \notin H$ for some $1 \leq i \leq \mu$. Then there exist k and ℓ such that (x_i, y_k) and (x_ℓ, y_i) form an exchange pair, where $k > \mu$ and $k \neq \ell$.*

Proof. Since $c_i \geq \mu+1$ there exists a k such that $k > \mu$ and $(x_i, y_k) \in H$. Further, $c_k \leq \mu$ and $c_i \geq \mu + 1$ imply that there exist an $\ell \neq k$ such that $(x_\ell, y_i) \in H$ and $(x_\ell, y_k) \notin H$. □

LEMMA 4.4. *Let H be any realization of (c, c) such that $(x_j, y_j) \in H$ for some $j > \mu$. Then there exist k and ℓ such that (x_j, y_j) and (x_ℓ, y_k) form an exchange pair, where $k \leq \mu$ and $k \neq \ell$.*

Proof. Since $c_j \leq \mu$ there exists a k such that $k \leq \mu$ and $(x_j, y_k) \notin H$. Further, $c_k \geq \mu + 1$ implies that there exists an $\ell \neq k$ such that $(x_\ell, y_k) \in H$ and $(x_\ell, y_j) \notin H$. □

LEMMA 4.5. *If (c, c) is a bipartite degree sequence then d is a degree sequence.*

Proof. Let H be any realization of (c, c). Assume that $(x_i, y_i) \notin H$ for some $1 \leq i \leq \mu$. Let k and ℓ be as defined in Lemma 4.3. Perform an exchange, by deleting the edges (x_i, y_k) and (x_ℓ, y_i) and adding the edges (x_i, y_i) and (x_ℓ, y_k). Observe that this process does not destroy any existing high-degree edges in H. We repeat this process until H contains all high-degree edges. Assume now that H contains a low-degree edge, say (x_j, y_j) for some $j > \mu$. Let k and ℓ be as defined in Lemma 4.4. Delete the edges (x_j, y_j) and (x_ℓ, y_k) and add the edges (x_j, y_k) and (x_ℓ, y_j). Observe that this process does not destroy any existing high-degree edges and does not create any new low-degree edges in H. We repeat this process until H contains no low-degree edges.

Define a digraph $\tilde{D} = (V, \tilde{E})$ on the vertex set $V = \{v_1, \ldots, v_n\}$, where $(v_i, v_j) \in \tilde{D}$ iff $(x_i, y_j) \in H$ and $(x_j, y_i) \notin H$. Then $d^-_{\tilde{D}}(v_i) = d^+_{\tilde{D}}(v_i) = d_i$ for all $1 \leq i \leq n$. By Lemma 2.4, we obtain a symmetric digraph $D = (V, E)$ such that $d^-_D(v_i) = d^+_D(v_i) = d_i$ for all i. Obtain an undirected graph G from D by replacing the two symmetric arcs (v_i, v_j) and (v_j, v_i) with the edge (v_i, v_j). Then G is a realization of d. □

Lemmas 4.2 and 4.5 imply Theorem 4.1. It is fairly easy to see that the proof of

Lemma 4.5 implies a simple sequential algorithm to realize d from any realization of (c, c). In Section 7, parallel algorithms to achieve the same objective will be presented.

5. Majorization and Unit Transformations. Throughout this section, let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be sequences of length n, where $a_1 \geq a_2 \geq \cdots \geq a_n \geq 0$ and $b_1 \geq b_2 \geq \cdots \geq b_n \geq 0$.

5.1. Majorization. Majorization has been studied, over several decades, in the theory of inequalities [22, 16]. It captures the intuitive notion that the components of a vector are "less nearly equal" than are the components of another vector. Formally, we say that *a majorizes b*, denoted by $a \succeq b$, if $\sum_{i=1}^{k} a_i \geq \sum_{i=1}^{k} b_i$ for $k = 1, ..., n$, with equality for $k = n$. For example, $(4, 2, 1, 0) \succeq (2, 2, 2, 1)$.

Majorization was used by economists in measuring inequality of incomes and in studying the principle of transfers (cf. [22]). If $a_i \geq a_j + 2$ for some i and j, we say that the sequence $c = (c_1, \ldots, c_n)$, defined by $c_i = a_i - 1, c_j = a_j + 1$ and $c_k = a_k$ for $k \neq i, j$, is obtained from a by a *unit transformation* from i to j. Clearly, $a \succeq c$. A classical result, known as the Muirhead Lemma (cf. [22]), states that the converse is also true: If $a \succeq b$ then a can be transformed to b by performing a finite number of (successive) unit transformations on a. The following lemma presents the details.

LEMMA 5.1. *Suppose that $a \succeq b$. Define a sequence $\delta = (\delta_1, \ldots, \delta_n)$ by $\delta_i = \max\{0, (a_i - b_i)\}$, and define $\Delta(a, b) = \sum_{i=1}^{n} \delta_i$. Then a can be transformed to b by performing $\Delta(a, b)$ unit transformations. Further, $\Delta(a, b)$ equals the minimum number of unit transformations required to transform a to b.*

Proof. If $a = b$ then $\Delta(a, b) = 0$. Assume that $a \neq b$ and let i be the smallest index such that $a_i \neq b_i$. Observe that $a_i > b_i$, since $a \succeq b$. Let $j > i$ be the smallest index such that $a_j < b_j$. Define c to be the sequence obtained from a by performing a unit transformation from i to j. Clearly, $a \succeq c \succeq b$. By repeating this process on the sequence c, we can obtain b.

To prove the second part of the lemma, observe that if c is any sequence obtained from a by a unit transformation, then $\Delta(c, b) \geq \Delta(a, b) - 1$. □

5.2. An Algorithm for Computing Unit Transformations. In this subsection, we discuss the computation of unit transformations that are required to transform a to b. The basic idea is to compute the numbers $t(i, j)$, for $1 \leq i, j \leq n$, so that a can be transformed to b by performing $t(i, j)$ unit transformations from position i to position j. We give an example to illustrate the idea.

Example 1: Let $a = (15, 12, 9, 8, 5, 4, 4, 0)$ and $b = (11, 10, 9, 9, 9, 3, 3, 3)$. Put $c = a - b = (4, 2, 0, -1, -4, 1, 1, -3)$ and define an array P (respectively, M) whose components are given by: $P_i = \max\{0, c_i\}$ (respectively, $M_i = \max\{0, -c_i\}$) for $1 \leq i \leq 8$, i.e., $P = (4, 2, 0, 0, 0, 1, 1, 0)$ and $M = (0, 0, 0, 1, 4, 0, 0, 3)$. Now, P_1 and M_4 are the left-most positive components of P and M, respectively. As $M_4 = 1 < P_1$, we subtract 1 from P_1 and M_4 and add 1 to $t(1, 4)$. Then P and M become (3,2,0,0,0,1,1) and (0,0,0,0,4,0,0,3), respectively. Now P_1 and M_5 are the left-most positive components of P and M and, as $P_1 = 3 < M_5$, we subtract 3 from P_1 and M_5 and add 3 to $t(1, 5)$. Then P and M become (0,2,0,0,0,1,1) and (0,0,0,0,1,0,0,3), respectively. We continue this process until all the components of P and M become 0's, and we get $t(1, 4) = 1$, $t(1, 5) = 3$, $t(2, 5) = t(2, 8) = t(6, 8) = t(7, 8) = 1$. □

A simple linear-time sequential algorithm, based on the procedure explained in the above example, for computing unit transformations is as follows. Compute the arrays P and M and then scan both the arrays from left to right, starting at the position $i = 1$ in P and $j = 1$ in M. In each step compute the appropriate $t(i, j)$ and

either increment i or j. Since the arrays P and M are scanned only once and the number of reported $t(i,j)$'s is linear, the algorithm runs in linear time.

A parallel implementation of this algorithm is presented in Algorithm 1. In Step 2 of the algorithm, $P'[i] := \sum_{k=1}^{i} P[i]$ for $1 \leq i \leq n$. Moreover the arrays P' and M' are obtained in the sorted order. After cross-ranking the elements of P' in M' and vice-versa, we know precisely the appropriate $t(i,j)$'s, for each value of i and j. For each value of i we can store the $t(i,j)$'s in an array, with respect to increasing j. Alternatively, we can store the $t(i,j)$'s in an $n * n$ matrix, without initializing the matrix, by a standard method (see [1]).

1. Compute $c_i := a_i - b_i$ for $1 \leq i \leq n$ and the arrays P and M.
2. Compute prefix sums of P and M and store them in the arrays P' and M', respectively.
3. Cross-rank the arrays P' and M' by the algorithm of [14] and then compute $t(i,j)$'s.

Algorithm 1: An algorithm for computing unit transformations.

THEOREM 5.2. *Let a and b be sequences of length n such that $a \succeq b$. Algorithm 1 computes the numbers $t(i,j)$, for $1 \leq i,j \leq n$, such that a can be transformed to b by performing $t(i,j)$ unit transformations from the position i to j in a. The algorithm runs in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. The proof of correctness is straightforward. To analyze the complexity, note that the prefix sums and the cross ranking of two sorted arrays can be computed in $O(\log n)$ time using $O(n/\log n)$ processors. □

5.3. Properties of Unit transformations. In this subsection we present properties of the numbers $t(i,j)$ computed by Algorithm 1. These properties will be used to design a parallel algorithm for realizing bipartite sequences.

Observe $t(i,j) = 0$ whenever $i \geq j$ or $a_j \geq b_j$. We need a few definitions to state additional properties of $t(i,j)$'s. For $1 \leq i \leq n$, define $A(i)$ to be the set of all j such that Algorithm 1 reports a unit transformation from i to j, i.e., $A(i) = \{j : t(i,j) \neq 0\}$ (see Example 2). Observe that $A(i) = \emptyset$ if and only if $a_i \leq b_i$. Similarly, define $B(j) = \{i : t(i,j) \neq 0\}$, and note that $B(j) = \emptyset$ if and only if $a_j \geq b_j$. Furthermore, if $k \in A(i)$ (respectively, $B(j)$), then $k > i$ (resp. $k < j$) and $a_k < b_k$ (respectively, $a_k > b_k$). The elements of $A(i)$ and $B(j)$ will always be listed in the increasing order.

Example 2: In Example 1,
$A(1) = \{4,5\}$, $A(2) = \{5,8\}$, $A(6) = A(7) = \{8\}$;
$B(4) = \{1\}$, $B(5) = \{1,2\}$, $B(8) = \{2,6,7\}$;
all other $A(i)$'s and $B(j)$'s are $\emptyset$. □

LEMMA 5.3.
1. *If $A(i) \neq \emptyset$, then $a_i = b_i + \sum_{j \in A(i)} t(i,j)$.*
2. *If $B(j) \neq \emptyset$, then $b_j = a_j + \sum_{i \in B(j)} t(i,j)$.*

Proof. Follows from the definition of $t(i,j)$'s in Algorithm 1. □

Let $\alpha(i) = \min\{A(i)\}$[2]. For $1 \leq i,j \leq n$, define (see Example 3)

$$\beta(i,j) = \begin{cases} 0 & \text{if } i > j \text{ or } a_j \geq b_j \\ a_j + \sum_{k \in B(j), k \geq i+1} t(k,j) & \text{otherwise.} \end{cases}$$

[2] By definition, $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$.

For $1 \leq i \leq n$, define

$$\gamma(i) = \begin{cases} \beta(i+1, \alpha(i)) & \text{if } A(i) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Example 3: In Example 2,
$\alpha(1) = 4$, $\alpha(2) = 5$, $\alpha(6) = \alpha(7) = 8$;
$\beta(3,5) = \beta(4,5) = \beta(5,5) = a_5 = 5$, $\beta(1,5) = a_5 + t(2,5) = 5 + 1 = 6$;
$\gamma(1) = \beta(2, \alpha(1)) = \beta(2,4) = \beta(3,4) = \beta(4,4) = a_4 = 8$. □

LEMMA 5.4. *Let j be an integer such that the elements of $B(j)$ are $i_1, i_2, \ldots, i_k$, where $k \geq 2$. Then $A(i_q) = \{j\}$ for all $2 \leq q \leq k-1$, and $\alpha(i_p) = j$ for all $2 \leq p \leq k$.*

Proof. Follows from definitions. □

LEMMA 5.5. *Let j be an integer such that the elements of $B(j)$ are $i_1, i_2, \ldots, i_k$, where $k \geq 2$. Then $\gamma(i_k) = a_j$ and $\gamma(i_p) = a_j + \sum_{q=p+1}^{k} t(i_q, j)$ for all $2 \leq p \leq k-1$. Furthermore, if $\alpha(i_1) = j$, then $\gamma(i_1) = \gamma(i_2) + t(i_2, j)$.*

Proof. By Lemma 5.4, $\gamma(i_k) = j$. So $\gamma(i_k) = \beta(i_k + 1, j)) = a_j$; the second equality follows from the fact that $t(\ell, j) = 0$ for $\ell \geq i_k + 1$. Consider now any p such that $2 \leq p \leq k-1$. We have

$$\begin{aligned} \gamma(i_p) &= \beta(i_{p+1} + 1, \alpha(i_p)) \\ &= \beta(i_{p+1} + 1, j) \qquad \text{(by Lemma 5.4)} \\ &= a_j + \sum_{q=i_{p+1}}^{k} t(i_q, j). \end{aligned}$$

The proof of the remaining part is similar. □

6. Realizing Bipartite Sequences. We present in this section a parallel algorithm for computing a bipartite graph that realizes a given pair of sequences. Throughout this section, $X = \{x_1, x_2, \ldots, x_m\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and r and s denote two nonnegative integer sequences, where $n \geq r_1 \geq r_2 \geq \cdots \geq r_m \geq 0$ and $m \geq s_1 \geq s_2 \geq \cdots \geq s_n \geq 0$. Given the pair (r, s), the problem is to compute a bipartite graph $H = (X, Y, E)$, if it exists, such that $d(x_i) = r_i$ and $d(y_j) = s_j$.

6.1. The Characterization. The following theorem, known as the Gale-Ryser theorem, characterizes bipartite sequences [13, 24, 11, 22]. Before stating the theorem, we require a definition. The *conjugate sequence* of r, denoted by $r^\star = (r_1^\star, \ldots, r_n^\star)$, is defined as $r_k^\star = |\{i : r_i \geq k\}|$ for $k = 1, \ldots, n$. Both r and $r^\star$ may be visualized by means of a (0,1)-matrix of size $m \times n$ in which the ith row contains r_i 1's left-justified; then $r_k^\star$ is the number of 1's in the kth column.

THEOREM 6.1 (THE GALE-RYSER THEOREM). *The pair (r, s) is a bipartite sequence if and only if $r^\star \succeq s$.*

Theorem 6.1 suggests a simple parallel algorithm to decide if (r, s) is a bipartite sequence. Moreover, a network-flow based proof of this theorem (cf. [11]), combined with the results of [20], leads to a randomized parallel algorithm to realize (r, s).

6.2. An Algorithm. We present in this section a parallel algorithm for computing a bipartite graph $H = (X \cup Y, E)$ that realizes the pair (r, s). It follows from the Gale-Ryser theorem that the pair $(r, r^\star)$ is a bipartite sequence. In the corresponding realization $F = (X, Y, E')$, the neighbors of y_j are $x_1, \ldots, x_p$, where $p = r_j^\star$. Represent $N_F(y_j)$ by the interval $[1, \ldots, p]$ of integers. Our algorithm is based on the following lemma.

LEMMA 6.2. *Suppose that (r, s) is a bipartite sequence. If the sequence t is obtained from s by a unit transformation, then (r, t) is also a bipartite sequence.*

Proof. Let $H = (X, Y, E)$ be a realization of (r, s) and t be obtained from s by a unit transformation from i to j. Since $s_i \geq s_j + 2$, there exists a neighbor, say x_k, of y_i that is not a neighbor of y_j. Let $H^{'}$ be obtained from H by deleting the edge (x_k, y_i) and adding the edge (x_k, y_j). Then $H^{'}$ is a realization of (r, t). □

We say that the graph $H^{'}$ constructed in the above proof is obtained from H by a *transfer of a neighbor* between x_i and x_j. The main steps in our algorithm for computing a realization of (r, s) are:

Step 1: Compute $r^{\star}$. Test if $r^{\star} \succeq s$. If not, declare that *(r,s) is not a bipartite sequence* and STOP.

Step 2: Obtain the realization $F = (X, Y, E^{'})$ of $(r, r^{\star})$ by computing $N_F(y_j) = [1, \ldots, r_j^{\star}]$.

Step 3: Transfer *appropriate neighbors* among the vertices of Y in F to obtain H.

The parallel implementation of Step 3, which is the difficult step, is based on the properties of the unit transformations (Section 5.3) required to transform the sequence $r^{\star}$ to s. For convenience of notation, we use the sequences a and b in place of $r^{\star}$ and s, respectively. Apply Algorithm 1 to the pair (a, b) and let $t(i, j)$, $1 \leq i, j \leq n$, be the unit transformations computed by the algorithm. Recall from Section 5.3 the definitions of $A(i)$, $B(j)$, $\alpha(i)$, $\beta(i, j)$ and $\gamma(i)$. Consider an i such that $A(i) \neq \emptyset$. We define intervals $T(i, j)$, where $j \in A(i)$, as follows. In the realization F of (r, a), all vertices x_p such that $p \in T(i, j)$ will be "transferred" from y_i to y_j to obtain the required bipartite graph H. Let the elements of $A(i)$ (in the increasing order) be $j_1 = \alpha(i), j_2, \ldots, j_k$. Define (see Figure 2)

$$T(i, j_1) = [\gamma(i) + 1, \ldots, \gamma(i) + t(i, j_1)] \text{ and}$$

$$T(i, j_p) = [a_i - \sum_{q=2}^{p} t(i, j_q) + 1, \ldots, a_i - \sum_{q=2}^{p-1} t(i, j_q)], \text{ for } 2 \leq p \leq k.$$

Observe that $|T(i, j_p)| = t(i, j_p)$ for all $1 \leq p \leq k$. We state in Algorithm 2 the procedure for realizing (r, s).

Example 4: Let $r = (4, 3, 3, 2, 2, 2, 1)$ and $b = s = (4, 3, 3, 3, 2, 2)$. Then
$X = \{x_1, \ldots, x_7\}$, $Y = \{y_1, \ldots, y_6\}$, and $a = r^{*} = (7, 6, 3, 1, 0, 0)$.
Applying Algorithm 1 to the pair (a, b) we get
$t(1, 4) = 2$, $t(1, 5) = 1$, $t(2, 5) = 1$, $t(2, 6) = 2$;
$A(1) = \{4, 5\}$, $A(2) = \{5, 6\}$;
$B(4) = \{1\}$, $B(5) = \{1, 2\}$, $B(6) = \{2\}$;
$\alpha(1) = 4$, $\alpha(2) = 5$, $\gamma(2) = a_5 = 0$, $\gamma(1) = a_4 = 1$.
The realization F of (r, a) has the following implicit representation. Recall that $(x_k, y_i) \in F$ iff $k \in N_F(y_i)$, and $N_F(y_i) = [1, a_i]$ is represented by its endpoints.
$N_F(y_1) = [1, 7]$, $N_F(y_2) = [1, 6]$, $N_F(y_3) = [1, 3]$,
$N_F(y_4) = [1, 1]$, $N_F(y_5) = N_F(y_6) = \emptyset$,
$T(1, 4) = [\gamma(1) + 1, \gamma(1) + t(1, 4)] = [2, 3]$,
$T(1, 5) = [a_1 - t(1, 5) + 1, a_1] = [7, 7]$,
$T(2, 5) = [\gamma(2) + 1, \gamma(2) + t(2, 5)] = [1, 1]$,
$T(2, 6) = [a_2 - t(2, 6) + 1, a_2] = [5, 6]$.

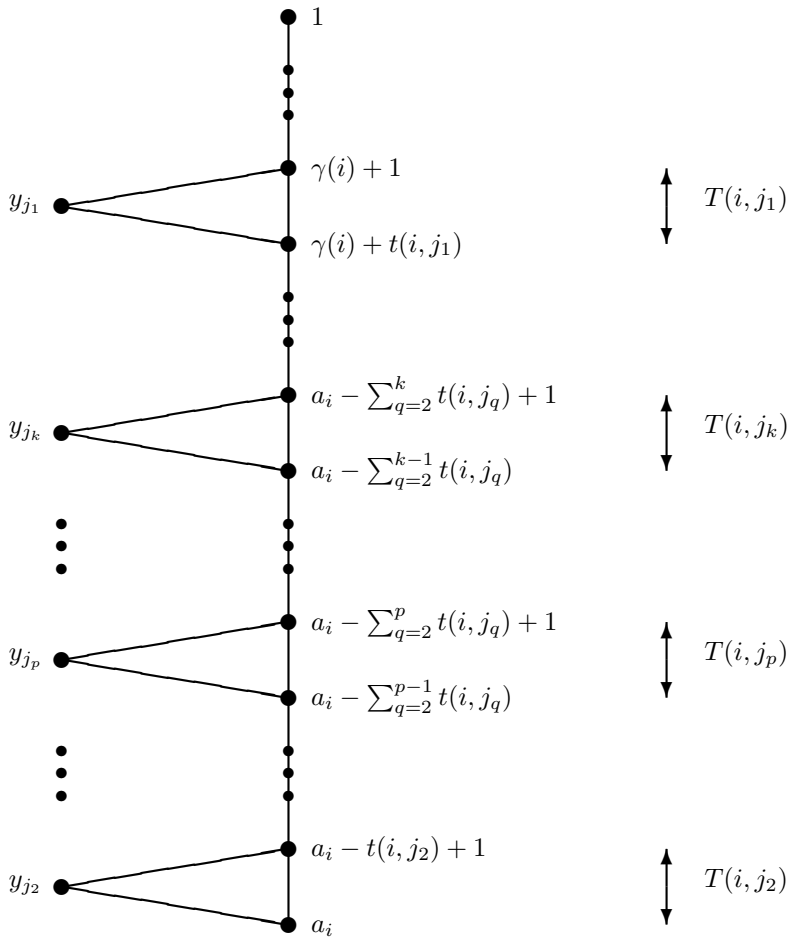

FIG. 2. *Defining $T(i,j)$. For example, $T(i,j_p)$ is the set of elements $a_i-\sum_{q=2}^{p}t(i,j_q)+1, a_i-\sum_{q=2}^{p}t(i,j_q)+2,\ldots,a_i-\sum_{q=2}^{p-1}t(i,j_q)$.*

The desired realization H of (r,s) has the following implicit representation.

$$N_H(y_1)=N_F(y_1)-(T(1,4)\cup T(1,5))=[1,1]\cup[4,6],$$
$$N_H(y_2)=N_F(y_2)-(T(2,5)\cup T(2,6))=[2,4],$$
$$N_H(y_3)=N_F(y_3)=[1,3],$$
$$N_H(y_4)=N_F(y_4)\cup T(1,4)=[1,3],$$
$$N_H(y_5)=N_F(y_5)\cup(T(1,5)\cup T(2,5))=[1,1]\cup[7,7],$$
$$N_H(y_6)=N_F(y_6)\cup T(2,6)=[5,6].\qquad \square$$

LEMMA 6.3. *The bipartite multigraph $H=(X\cup Y,E)$ computed in Algorithm 2 is a simple bipartite graph.*

Proof. It suffices to show that there are no multiple edges in H that are incident to any vertex y_j. This is equivalent to proving that $N_H(y_j)$ is a set (i.e., it has no duplicate elements). If $a_j\geq b_j$, $N_H(y_j)$ is a set as $N_F(y_j)$ is one. Consider now the case that $a_j<b_j$. Then $N_H(y_j)=N_F(y_j)\cup J$, where $J=\bigcup_{i\in B(j)}T(i,j)$. We prove that $N_H(y_j)$ has no duplicates by showing that J has no duplicates and that

Input: A pair (r, s) of sequences.
Output: A bipartite graph $H = (X, Y, E)$, if it exists. H is specified implicitly by the neighborhoods of vertices in Y.

1. Compute $r^\star$. Test if $r^\star \succeq s$. If the test fails, then declare that *(r,s) is not a bipartite sequence* and STOP.
2. Set $(a, b) := (r^\star, s)$ and apply Algorithm 1 to compute the unit transformations $t(i, j)$ for $1 \le i, j \le n$.
3. Compute the bipartite graph F which realizes (r, a). F is represented implicitly: for each $1 \le j \le n$, compute the set of neighbors of y_j, namely $N_F(y_j) = [1, \ldots, a_j]$.
4. Compute $A(i)$ and $B(j)$ for $1 \le i, j \le n$.
5. Compute $\beta(i, j)$ for all i and j such that $t(i, j) \neq 0$; compute $\gamma(i)$ for $1 \le i \le n$.
6. Compute the intervals $T(i, j)$ for $1 \le i \le n$ and $j \in A(i)$.
7. For all i such that $a_i > b_i$, compute $N_H(y_i) := N_F(y_i) - \bigcup_{j \in A(i)} T(i, j)$.
8. For all j such that $a_j < b_j$, compute $N_H(y_j) := N_F(y_j) \cup (\bigcup_{i \in B(j)} T(i, j))$.
9. For all i such that $a_i = b_i$, set $N_H(y_i) := N_F(y_i)$.

Algorithm 2: Algorithm for computing a bipartite graph.

$N_F(y_j) \cap J = \emptyset$. Recall that $N_F(y_j) = [1, \ldots, a_j]$.

Let the elements of $B(j)$ (in the increasing order) be $i_1, \cdots, i_k$. Since $j \in A(i_1)$, we have $\alpha(i_1) \le j$. First consider the case $\alpha(i_1) = j$.

Case 1: $k = 1$. Then $J = T(i_1, j) = [\gamma(i_1) + 1, \ldots, \gamma(i_1) + t(i_1, j)] = [a_j + 1, \ldots, a_j + t(i_1, j)]$. Thus J is a set and $N_F(y_j) \cap J = \emptyset$.

Case 2: $k \ge 2$. By Lemma 5.4, $\alpha(i_q) = j$ for $2 \le q \le k$. By definition, $T(i_q, j) = [\gamma(i_q) + 1, \ldots, \gamma(i_q) + t(i_q, j)]$ for $1 \le q \le k$. Then, by Lemma 5.5, $T(i_q, j) = [\gamma(i_q) + 1, \ldots, \gamma(i_{q-1})]$ for $2 \le q \le k$ and $T(i_1, j) = [\gamma(i_1) + 1, \ldots, \gamma(i_1) + t(i_1, j)]$. Hence, for distinct p and q such that $1 \le p, q \le k$, we obtain $T(i_p, j) \cap T(i_q, j) = \emptyset$ and thus J has no duplicates. Further, $J = \cup_{p=1}^{k} T(i_p, j) = [\gamma(i_k) + 1, \ldots, \gamma(i_k) + \sum_{p=1}^{k} t(i_p, j)]$, and so $\min(J) = \gamma(i_k)$. We now obtain $N_F(y_j) \cap J = \emptyset$, as $\max(N_F(j)) = a_j = \gamma(i_k) = \min(J) - 1$ (the second equality follows from Lemma 5.5).

Now consider the other case, namely $\alpha(i_1) < j$. Let $j_1 = \alpha(i_1), j_2, \ldots, j_p = j$ be the elements of $A(i_1)$ that are $\le j$. By definition

$$T(i_1, j) = [a_{i_1} - \sum_{q=2}^{p} t(i_1, j_q) + 1, \ldots, a_{i_1} - \sum_{q=2}^{p-1} t(i_1, j_q)].$$

Case 1: $k = 1$. Then $J = T(i_1, j)$ and

$$\begin{aligned}
\max(N_F(j)) &= a_j \\
&< b_j \\
&\le b_{i_1} \qquad \text{(as } i_1 \le j \text{ and } b \text{ is nonincreasing)} \\
&= a_{i_1} - \sum_{u \in A(i_1)} t(i_1, u) \qquad \text{(by Lemma 5.3)} \\
&< a_{i_1} - \sum_{q=2}^{p} t(i_1, j_q) \\
&= \min(J) - 1.
\end{aligned}$$

Hence $N_H(y_j) \cap J = \emptyset$.

Case 2: $k \geq 2$. Put $I = \bigcup_{p=2}^{k} T(i_p, j)$; then $J = T(i_1, j) \cup I$. We claim that I is a set. By Lemma 5.4, $\alpha(i_p) = j$ for all $2 \leq p \leq k$. By Lemma 5.5, $T(i_p, j) = [\gamma(i_p) + 1, \ldots, \gamma(i_{p-1})]$ for $3 \leq p \leq k$, and $T(i_2, j) = [\gamma(i_2) + 1, \ldots, \gamma(i_2) + t(i_2, j)]$. Using $\gamma(i_k) = a_j$, we obtain $I = [a_j + 1, \ldots, \gamma(i_2) + t(i_2, j)]$, completing the claim. Now,

$$\begin{aligned}
\max(I) &= \gamma(i_2) + t(i_2, j) \\
&= a_j + \sum_{q=3}^{k} t(i_q, j) + t(i_2, j) \qquad \text{(by Lemma 5.5)} \\
&< a_j + \sum_{u \in B(j)} t(u, j) \\
&= b_j \qquad \text{(by Lemma 5.3)} \\
&\leq b_{i_1} \qquad \text{(as } i_1 \leq j \text{ and } b \text{ is nonincreasing)} \\
&= a_{i_1} - \sum_{u \in A(i_1)} t(i_1, u) \qquad \text{(by Lemma 5.3)} \\
&< a_{i_1} - \sum_{q=2}^{p} t(i_1, j_q) \\
&= \min(T(i_1, j)) - 1.
\end{aligned}$$

Hence $T(i_1, j) \cap I = \emptyset$, implying that J has no duplicates. Finally, $\max(N_F(y_j)) = a_j = \min(I) - 1 = \min(J) - 1$ and hence $N_F(y_j) \cap J = \emptyset$. □

LEMMA 6.4. *In the graph $H = (X \cup Y, E)$ computed by Algorithm 2, $d_H(x_i) = r_i$ and $d_H(y_j) = b_j$ $(= s_j)$.*

Proof. Clearly, $d_H(x_i) = d_F(x_i) = r_i$. If $a_j = b_j$ then $d_H(y_j) = b_j$, since $N_H(y_j) = N_F(y_j)$ and $d_F(y_j) = a_j$. Suppose next that $a_j < b_j$. Then

$$\begin{aligned}
d_H(y_j) &= |N_F(y_j)| \\
&= |N_F(y_j)| + \sum_{i \in B(j)} |T(i, j)| \\
&= a_j + \sum_{i \in B(j)} t(i, j) \\
&= b_j.
\end{aligned}$$

Consider now the remaining case: $a_j > b_j$. Let $i = j$ for convenience and put $I = \sum_{u \in A(i)} T(i, u)$. Then $N_H(y_i) = N_F(y_i) - I$. Let the elements of $A(i)$ be $j_i = j, j_2, \ldots, j_k$. Now

$$\begin{aligned}
\gamma(i) + t(i, j_1) &\leq a_{j_1} + \sum_{q \in B(j_1)} t(q, j_1) \\
&= b_{j_1} \\
&\leq b_i \\
&= a_i - \sum_{q=1}^{k} t(i, j_q)
\end{aligned}$$

$$< a_i - \sum_{q=2}^{k} t(i, j_q).$$

Thus $I = [\gamma(i) + 1, \ldots, \gamma(i) + t(i, j_1)] \cup [a_i - 1, \ldots, a_i - \sum_{q=2}^{k} t(i, j_q)]$ and hence $I \subseteq N_F(y_i)$. Now

$$\begin{aligned} |I| &= t(i, j_1) + \sum_{q=2}^{k} t(i, j_q) \\ &= \sum_{u \in A(i)} t(i, u) \\ &= a_i - b_i. \end{aligned}$$

Finally, $d_H(y_i) = |N_H(y_i)| = |N_F(y_i)| - |I| = a_i - (a_i - b_i) = b_i$. □

We now outline the parallel complexity of Algorithm 2. As mentioned in Introduction, the graphs computed by our algorithms possess implicit representations: the graphs can be stored in $O(n)$ space. For ease of notation, we assume that n denotes the total number of components of r and s (i.e., $n := n + m$). To analyze Step 1, we first sort the sequences r and s using the algorithm of [8] and store the sorted sequences in the arrays R and S, respectively. Sorting can be performed in $O(\log n)$ time using $O(n)$ EREW PRAM processors. We show how to perform all other steps in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. $r^\star$ can be computed by cross-ranking [14] the array R with the array $(1, 2, \ldots, n)$. Step 2 can be performed by using Algorithm 1. It reports $t(i, j)$'s in a lexicographic order. Steps 4, 5, and 6 can be implemented by performing the appropriate prefix sums. In Step 7, observe that $\bigcup_{j \in A(i)} T(i, j)$ is union of at most three intervals, and hence can be computed within the same resource bounds. Therefore, Algorithm 2 can be implemented in $O(\log n)$ time using $O(n)$ EREW PRAM processors. We summarize the result in the following.

THEOREM 6.5. *Given a pair (r, s) of nonnegative integer sequences, a bipartite graph that realizes (r, s) can be computed in $O(\log n)$ time using $O(n)$ EREW PRAM processors, where n is the total number of components of r and s. Moreover, if the sequences r and s are given as sorted sequences, then the graph can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

7. Realizing Degree Sequences. Let $d = (d_1, \ldots, d_n)$ be a nonnegative integer sequence, where $n > d_1 \geq d_2 \geq \ldots \geq d_n \geq 0$. In this section we present parallel algorithms to compute a graph $G = (V, E)$ on the vertex set $V = \{v_1, \ldots, v_n\}$ that realizes d. Our parallel algorithms are based on the proof of Theorem 4.1 and the main steps are:

Step 1: From d, compute the sequence (c, c) and a realization $H = (X, Y, E)$ of (c, c). From H compute another realization H' of (c, c) such that $(x_i, y_i) \in H'$ if and only if $1 \leq i \leq \mu$.

Step 2: Compute from H' a symmetric digraph $D = (V, E)$ such that $d_D^-(v_i) = d_D^+(v_i) = d_i$. Then compute G.

We first discuss the parallel implementation of Step 2; the essential ideas are described in the proof of Lemma 2.4. Compute the digraph $\tilde{D} = (V, \tilde{E})$ on the vertex set $V = \{v_1, \ldots, v_n\}$, where $(v_i, v_j) \in \tilde{D}$ if and only if $(x_i, y_j) \in H'$ and $(x_j, y_i) \notin H'$. Then, the indegree of each vertex in $\tilde{D}$ is same as its outdegree. Compute Eulerian tours in $\tilde{D}$ using the algorithm of [5]; this algorithm computes an Eulerian tour for each connected component of $\tilde{D}$. Each trail is stored in an array. Using parallel

list ranking, compute the number of edges in each trail. Consider first the even trails. Using the array representation of the trails, delete alternate arcs and insert the appropriate arcs as stated in the proof of Lemma 2.4. After this step, we are left with an even number of odd trails and we group them in pairs. For each pair, determine if it belongs to Case 1 or Case 2 in the proof of Lemma 2.4. In either case, delete and insert appropriate arcs. We now have the required symmetric digraph $D = (V, E)$. Compute an undirected graph G from D by replacing the two symmetric arcs (v_i, v_j) and (v_j, v_i) with the edge (v_i, v_j). Then G is the desired realization of d.

We now analyze the complexity of Step 2. The graph $H^{'}$ is represented in terms of adjacency lists: for each $y \in Y$, we maintain $N_{H'}(y)$ as a list. Let m denote the number of edges of G. The digraph $\tilde{D}$ can be computed in $O(\log n)$ time using $O(m)$ CRCW processors as follows. For each vertex, sort the vertices adjacent to it by the algorithm of [8], and test in $O(\log n)$ time if $(x_j, y_i) \in H$. The algorithm of [5] computes Eulerian tours in $\tilde{D}$ in $O(\log n)$ time using $O(m)$ CRCW PRAM processors. Parallel list ranking requires $O(\log n)$ time using $O(m)$ processors and the lists can be represented as consecutive elements in an array. Once we have the array representation, all other steps can be performed within the claimed complexity bounds. Hence the overall complexity of performing Step 2 is $O(\log n)$ time using $O(m)$ CRCW PRAM processors.

In the remainder of this section, we discuss the parallel implementation of Step 1. Recall from Section 4 the definitions of high-degree and low-degree edges. In Step 1, we wish to compute a realization of (c, c) which has all high-degree edges and no low-degree edges. The main steps involved in obtaining such a realization are (i) compute a bipartite realization H of (c, c) using, for example, Algorithm 2. (ii) compute appropriate exchange pairs in H to obtain all high-degree edges and (iii) compute appropriate exchange pairs in the resulting graph to remove the low-degree edges.

We provide two alternate parallel algorithms for implementing Step 1. The first algorithm is simple and intuitive and is based on several interesting lemmas. It has high complexity, since it proceeds by reducing the computation to that of recursively computing maximal matchings. On the other hand, the second algorithm is efficient, though involved: it is based on the implicit structure of H computed by Algorithm 2. We present the first algorithm Section 7.1 and the second one in Section 7.2.

7.1. The First Algorithm. Recall that H is a realization of (c, c). If there are missing high-degree edges in H, then we execute the procedure described in Section 7.1.1. This is followed by executing the procedure for low-degree edges, as described in Section 7.1.2. After performing these two steps, we are left with a bipartite realization H' of H, which has all high-degree edges and no low-degree edges. This gives the outline of the first algorithm and we summarize the result in the following.

THEOREM 7.1. *Given an integer sequence d, a realization of d can be computed in $O(\log^4 n)$ time using $O(n^3)$ CRCW PRAM processors.*

Proof. The proof and the complexity analysis follow from Lemmas 2.4, 7.4, and 7.5. □

7.1.1. Procedure for High-Degree Edges. In this subsection, we present a parallel algorithm to compute appropriate exchange pairs in the realization H, so that by performing the corresponding exchanges (see Figure 1) we obtain another realization of (c, c) having all high-degree edges. The procedure has two phases.

In the first phase, we restrict ourselves to the subgraph of H induced by the high-degree vertices. Compute a maximal number of exchange pairs in this subgraph

as follows. Define a graph G' on the vertex set $\{1, 2, ..., \mu\}$. There is an edge between i and j in G' if and only if $(x_i, y_i), (x_j, y_j) \notin H$ and the edges (x_i, y_j) and (x_j, y_i) form an exchange pair. Compute a maximal matching M in G' and then perform the corresponding exchanges in H. For notational simplicity, let H be the resulting bipartite graph after performing these exchanges.

In the second phase, we obtain the rest of the missing high-degree edges in H. Define as follows a bipartite graph $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ over the missing high-degree edges and the corresponding exchange pairs in H. $\mathcal{P} = \{p_i : (x_i, y_i) \notin H, 1 \leq i \leq \mu\}$ and $\mathcal{Q} = \{q_{lk} : (x_l, y_k) \notin H, l \neq k \text{ and } n \geq k > \mu\}$. Further, $(p_i, q_{lk}) \in \mathcal{E}$ if and only if the edges (x_i, y_k) and (x_l, y_i) form an exchange pair in H. The following lemma establishes an important property of the bipartite graph $\mathcal{H}$.

LEMMA 7.2. *Suppose that there is at least one high-degree edge missing in H and let the bipartite graph $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ be as defined above. Then (i) $d(p_i) \geq 1$ for all i and (ii) if $(p_i, q_{lk}) \in \mathcal{E}$, then $d(p_i) \geq d(q_{lk})$, where $d(z)$ is the degree of vertex z in $\mathcal{H}$.*

Proof. Part (i) follows from Lemma 4.3. We say a vertex x_i is a *high-degree neighbor* of y_k, if $(x_i, y_k) \in H$ and $1 \leq i \leq \mu$. Let A_1 be the set of high-degree neighbors of y_k, which are not neighbors of y_i in H. Let A_2 be the set of high-degree neighbors of y_k which are also neighbors of y_i in H. Observe that $d(q_{lk}) \leq |A_1| + |A_2|$. To prove part (ii) of the lemma, we show that $d(p_i) \geq |A_1| + |A_2|$. This is achieved by proving that $N_{\mathcal{H}}(p_i) \supseteq B_1 \cup B_2$, where $|B_1| \geq |A_1|$, $|B_2| \geq |A_2|$ and $B_1 \cap B_2 = \emptyset$. Recall that c_i is the degree of y_i in H and that $c_i > \mu \geq c_k$ (as $i \leq \mu$ and $k > \mu$). This implies that $|A_1^{'}| \geq |A_1|$, where $A_1^{'} = \{\alpha : x_\alpha \in N_H(y_i) - N_H(y_k), \alpha \neq k\}$. The edges (x_i, y_k) and (x_α, y_i) form an exchange pair for every $\alpha \in A_1^{'}$ and we define $B_1 = \{q_{\alpha k} : \alpha \in A_1^{'}\}$. Observe that $B_1 \subseteq N_{\mathcal{H}}(p_i)$. We define B_2 now. If $(x_j, y_i) \in H$ for some $1 \leq j \leq \mu$ and $(x_j, y_j) \notin H$, then $(x_i, y_j) \notin H$ by the fact that a maximal matching is found in Phase 1 of the algorithm. Hence $|A_2^{'}| \geq |A_2|$, where $A_2^{'} = \{\beta : (x_i, y_\beta) \in H, \beta > \mu, \beta \neq k\}$. For every $\beta \in A_2^{'}$, there exists a vertex, say $x_{\beta'}$, such that $\beta^{'} \neq \beta$ and (x_i, y_β) and $(x_{\beta'}, y_i)$ form an exchange pair in H. Define $B_2 = \{q_{\beta'\beta} : \beta \in A_2^{'}\}$ and note that $B_2 \subseteq N_{\mathcal{H}}(p_i)$. To complete the proof, observe that $B_1 \cap B_2 = \emptyset$. □

COROLLARY 7.3. *In $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$, there exists a matching that matches all vertices of $\mathcal{P}$.*

Proof. Follows from Lemmas 2.3 and 7.2. □

We discuss some algorithmic aspects of Lemma 7.2. We compute the bipartite graph $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ from H. From Corollary 7.3, we know that any maximum matching in $\mathcal{H}$ matches all vertices in $\mathcal{P}$. By finding a maximum matching in $\mathcal{H}$, we can compute the corresponding exchange pairs in H and hence obtain all the missing high-degree edges. Unfortunately, only randomized parallel algorithms are known for computing a maximum matching [19]. In order to solve our problem deterministically, we resort to the special structure of $\mathcal{H}$ stated in Lemma 7.2 and Corollary 7.3. The solution is presented in Algorithm 3.

LEMMA 7.4. *Algorithm 3 computes a bipartite realization of (c, c) having all high-degree edges. It runs in $O(\log^4 n)$ time using $O(n^3)$ CRCW PRAM processors.*

Proof. We first show the correctness of the algorithm. In every step, observe that (c, c) is the degree sequence of the bipartite graph H. Consider an iteration of the while loop. The correctness of Phase 1 is obvious. Assume that H has some high-degree edges missing after Phase 1 and let $\mathcal{H}$ be as defined in Phase 2. Then $\mathcal{E}$ is not empty by condition (i) of Lemma 7.2. Thus H has more high-degree edges at the end of Phase 2 than it has in the beginning.

While there are missing high-degree edges in H **do**
begin
Phase 1:

1. Compute a graph G' on the vertex set $\{1, 2, ..., \mu\}$; there is an edge between i and j if and only if (x_i, y_j) and (x_j, y_i) form an exchange pair.
2. Compute a maximal matching M in G' using the algorithm of [17].
3. From M, compute the exchange pairs in H and perform exchanges to obtain the corresponding high-degree edges; let the resulting graph be H.

Phase 2:

1. Compute a bipartite graph $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ from H, where $\mathcal{P} = \{p_i : (x_i, y_i) \notin H, 1 \leq i \leq \mu\}$, $\mathcal{Q} = \{q_{lk} : (x_l, y_k) \notin H, l \neq k \text{ and } n \geq k > \mu\}$, and $(p_i, q_{lk}) \in \mathcal{E}$ if and only if the edges (x_i, y_k) and (x_l, y_i) form an exchange pair in H.
2. Compute a maximal matching M in $\mathcal{H}$ using the algorithm of [17].
3. From M, compute the exchange pairs in H and perform exchanges to obtain the corresponding high-degree edges; let the resulting graph be H.

end.

Algorithm 3: A parallel algorithm to obtain the missing high-degree edges in H

We now discuss the complexity of the algorithm. By Lemmas 2.2 and Corollary 7.3, the maximal matching computed in Step 2 of Phase 2 has size at least $\frac{1}{2}|\mathcal{P}|$. Thus in each iteration of the while loop, the number of missing high-degree edges in H drops down by at least a factor of 2, which implies that the while loop is executed for at most $O(\log n)$ times. In each iteration, the graph G' can be computed in $O(1)$ time using $O(n^2)$ processors and the graph $\mathcal{H}$ in $O(1)$ time using $O(n^3)$ processors. Computing maximal matchings in G' and $\mathcal{H}$ takes $O(\log^3 n)$ time using $O(n^3)$ CRCW PRAM processors. Hence the overall complexity is as stated in the lemma. □

7.1.2. Procedure for Low-Degree Edges. Let H be the bipartite graph computed by Algorithm 3; H has all high-degree edges and possibly some low-degree edges. Our aim in this subsection is to transform H into another bipartite realization H' of (c, c) such that H' has all high degree-edges and no low-degree edges. Our algorithm for achieving this is similar to Algorithm 3, and we present below the necessary modifications.

As before, there are two phases. In the first phase, we restrict ourselves to the subgraph of H induced by the low-degree vertices. Define a graph G' on the vertex set $\{\mu + 1, ..., n\}$. There is an edge between i and j in G' if and only if the edges (x_i, y_i) and (x_j, y_j) form an exchange pair. In the second phase, define as follows a bipartite graph $\mathcal{H} = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ over the low-degree edges and the corresponding exchange pairs in H. $\mathcal{P} = \{p_i : (x_i, y_i) \in H, \mu < i \leq n\}$, and $\mathcal{Q} = \{q_{lk} : (x_l, y_k) \in H, l \neq k$ and $1 \leq k \leq \mu\}$. Further, $(p_i, q_{lk}) \in \mathcal{E}$ if and only if the edges (x_i, y_i) and (x_l, y_k) form an exchange pair in H. The results of Lemma 7.4 hold with this definition of $\mathcal{H}$ also. The rest of the discussion is similar to that of previous subsection. We conclude with the following.

LEMMA 7.5. *A bipartite realization of (c, c) having all high-degree edges and no low-degree edges can be computed in $O(\log^4 n)$ time using $O(n^3)$ CRCW PRAM processors.*

7.2. An Efficient Algorithm. We present an efficient computation of the bipartite graph H' introduced at the beginning of this section. The computation is

based on the structure of the bipartite graph computed by Algorithm 2.

The main result of this section is the following.

THEOREM 7.6. *Given an integer sequence d of length n, a graph that realizes d can be computed in $O(\log n)$ time using $O(n+m)$ CRCW PRAM processors, where m is the number of edges in the realization.*

7.2.1. The Structure. We present in this section the structure of the bipartite graphs F and H computed by Algorithm 2.

Both graphs F and H (defined on the vertex set $X \cup Y$) are specified by giving the neighbors of the vertices in Y. For $y_i \in Y$, $N_F(y_i)$ is an interval of vertices, namely $N_F(y_i) = \{x_k : 1 \leq k \leq a_i\}$, and we denote $N_F(y_i)$ by the interval $[1, \ldots, a_i]$. $N_H(y_i)$ is a union of at most two disjoint intervals of vertices (cf. the proof of Lemma 6.3): $N_H(y_i) = \{x_k : k \in (L_i \cup M_i)\}$, where L_i and M_i are defined below; we denote $N_H(y_i)$ by $L_i \cup M_i$. Recall the definitions of $t(i,j)$, $A(i)$, $B(j)$, $\alpha(i)$, $\beta(i,j)$, $\gamma(i)$ and $T(i,j)$ from Section 6. To define L_i and M_i, we distinguish between the following cases.

Case 1: $a_i = b_i$. Then $L_i = [1, \ldots, a_i]$ and $M_i = \emptyset$.

Case 2: $a_i > b_i$. Let the elements of $A(i)$ be $j_1, \ldots, j_k$. Then $L_i = [1, \ldots, \gamma(i)]$, and $M_i = [\gamma(i) + t(i, j_1) + 1, \ldots, a_i - \sum_{q=2}^{k} t(i, j_q)]$.

Case 3: $a_i < b_i$. Put $j = i$ and let the elements of $B(j)$ be $i_1, \ldots, i_k$.

Case 3.1: $\alpha(i_1) = j$. Then $L_j = [1, \ldots, a_j + \sum_{q=1}^{k} t(i_q, j)]$ and $M_j = \emptyset$.

Case 3.2: $\alpha(i_1) < j$. Let $j_1 = \alpha(i_1), j_2, \ldots, j_p = j$ be the elements of $A(i_1)$ that are $\leq j$. Then $L_j = [1, \ldots, a_j + \sum_{q=2}^{k} t(i_q, j)]$ and $M_i = [a_{i_1} - \sum_{q=2}^{p} t(i_1, j_q) + 1, \ldots, a_{i_1} - \sum_{q=2}^{p-1} t(i_1, j_q)]$.

Example 5: In Example 4, $L_1 = [1,1]$, $M_1 = [4,6]$; $L_2 = \emptyset$, $M_2 = [2,4]$; $L_3 = [1,3]$, $M_3 = \emptyset$; $L_4 = [1,3]$, $M_4 = \emptyset$; $L_5 = [1,1]$, $M_5 = [7,7]$; $L_6 = \emptyset$, $M_6 = [5,6]$. □

LEMMA 7.7. *For all $1 \leq i \leq n$, $\max(L_i) < \min(M_i)$.*

Proof. The neighborhoods $N_H(y_i)$ are defined in the proof of Lemma 6.3. It is routine to verify that $\max(L_i) < \min(M_i)$ holds in all the cases considered in that proof. □

7.2.2. The Algorithm. Our algorithm computes four bipartite graphs, namely $H_0, H_1, H_2, H_3 = H'$, on the same vertex set $X \cup Y$. These graphs are represented implicitly using the neighborhoods of vertices in Y. Throughout, $N_i(y)$ denotes the set of neighbors of $y \in Y$ in H_i, where $i \in \{0,1,2,3\}$.

The algorithm (Algorithm 4) has three steps. In the first step, we compute H_0 and H_1; they are the realizations of $(c, c^\star)$ and (c, c), respectively.

In the second step, we compute H_2 that has all high-degree edges, i.e., $(x_i, y_i) \in H_2$ for all $1 \leq i \leq \mu$; no new low-degree edges are created in this step. Suppose the high-degree edge (x_i, y_i) is absent in H_1. To create the edge (x_i, y_i) by performing an exchange, we need to find vertices x_l, y_k such that (x_i, y_k) and (x_l, y_i) form an exchange pair. To make sure that no multiple edges are created when parallel exchanges are done, we select $k = \alpha(i)$ (see Lemma 7.11). Further, we impose the condition $l \neq k$ in order to avoid creating any low-degree edges. More precisely, we choose $l = \theta(i) = \min\{\ell : \ell \neq \alpha(i), x_\ell \in N_1(y_i) - N_1(y_{\alpha(i)})\}$. See Step 2 of Algorithm 4.

In the last step, we compute H_3 by deleting from H_2 all low-degree edges without destroying the high-degree edges. Suppose the low-degree edge (x_j, y_j) is present in H_2. To destroy the edge (x_j, y_j) by performing an exchange, we need to find vertices x_l, y_k such that (x_j, y_j) and (x_l, y_k) form an exchange pair. To make sure that no

multiple edges are created when parallel exchanges are done, we select $k = \tau(j) = i$, where $j \in T(i,j)$ (see Lemma 7.14). Further, we impose the condition $l \neq k$ in order to avoid destroying the high-degree edges. More precisely, we choose $l = \psi(j) = \min\{\ell : \ell \neq \tau(j), x_\ell \in N_2(y_{\tau(j)}) - N_2(y_j)\}$. See Step 3 of Algorithm 4.

1. Let H_0 and H_1, respectively, be the realizations of $(c, c^\star)$ and (c, c) computed by Algorithm 2.
2. Compute $I = \{i : i \leq \mu, (x_i, y_i) \notin H_1\}$. Compute a new bipartite graph H_2 from H_1 by doing in parallel the following for all $i \in I$: Delete the edges $(x_i, y_{\alpha(i)})$ and $(x_{\theta(i)}, y_i)$ and add the edges (x_i, y_i) and $(x_{\theta(i)}, y_{\alpha(i)})$.
3. Compute $J = \{j : j > \mu, (x_j, y_j) \in H_2\}$. Compute a new bipartite graph H_3 from H_2 by doing in parallel the following for all $j \in J$: Delete the edges (x_j, y_j) and $(x_{\psi(j)}, y_{\tau(j)})$ and add the edges $(x_{\psi(j)}, y_j)$ and $(x_j, y_{\tau(j)})$.
4. Output H_3.

Algorithm 4: An efficient algorithm.

The rest of this section is devoted to prove the following important result.

LEMMA 7.8. *Algorithm 4 computes a realization of (c, c) that has all high-degree edges and no low-degree edges. It runs in $O(\log n)$ time using $O(n)$ EREW PRAM processors.*

We let, for ease of notation, $a = c^\star$ and $b = c$. Recall from Section 6 that the condition $k \in T(i,j)$ means: 'y_j gets x_k from y_i in H_0', i.e., $(x_k, y_i) \in H_0 - H_1$ and $(x_k, y_j) \in H_1 - H_0$.

The following two lemmas are used to prove the correctness of Step 2 of Algorithm 4.

LEMMA 7.9. *Let $i \leq \mu$ be such that $(x_i, y_i) \notin H_1$. Then there exists a unique j such that $i \in T(i,j)$. Furthermore, $j = \alpha(i)$ and $j > \mu$.*

Proof. The condition $i \leq \mu$ implies that $a_i \geq \mu$, and thus $(x_i, y_i) \in H_0$. Further, the condition $(x_i, y_i) \notin H_1$ implies that $a_i > b_i$ as $N_1(y_i) \subseteq N_0(y_i)$ otherwise. Let the elements of $A(i)$ be $j_1, \ldots, j_k$. The condition $(x_i, y_i) \notin H_1$ implies that there exists a j_ℓ such that $i \in T(i, j_\ell)$. From the definition of $T(i,j)$ it follows that j_ℓ is unique. We now show that j_ℓ satisfies the other properties of j stated in the lemma. Suppose for contradiction that $j_\ell \neq \alpha(i)$, i.e., $j_\ell > \alpha(i)$. Then $T(i, j_\ell) = [a_i - \sum_{q=2}^{\ell} t(i, j_q) + 1, \ldots, a_i - \sum_{q=2}^{\ell-1} t(i, j_q)]$, and

$$\begin{aligned}
b_i &= |N_1(y_i)| \\
&= |L_i| + |M_i| \\
&\leq \max(M_i) \qquad \text{(by Lemma 7.7)} \\
&= a_i - \sum_{q=2}^{k} t(i, j_q) \\
&\leq a_i - \sum_{q=2}^{\ell} t(i, j_q) \\
&< \min(T(i, j_\ell)) \\
&\leq i \qquad (\text{as } i \in T(i, j_\ell)) \\
&\leq \mu,
\end{aligned}$$

which contradicts $b_i \geq \mu + 1$. To prove the remaining part, observe that $\gamma(i) \geq a_{j_\ell}$

and that i belongs to $T(i, j_\ell)$ only if $\gamma(i) < i$; the last inequality is possible only if $j_\ell > \mu$ (as $a_{j_\ell} \geq \mu + 1$ if $j_\ell \leq \mu$). □

LEMMA 7.10. *Let $\mu < j \leq n$. There exists at most one i such that $i \in T(i, j)$. In that case, $i \leq \mu$ and $\alpha(i) = j$.*

Proof. That there exists at most one i is clear from the definition of $T(i, j)$ (cf. Figure 2). Observe that i belongs to $T(i, j)$ only if $i \leq \mu$. Then $\alpha(i) = j$ by Lemma 7.9. □

LEMMA 7.11. *The bipartite multigraph H_2 computed in Step 2 of Algorithm 4 is a simple bipartite graph that realizes (c, c). Moreover, $(x_i, y_i) \in H_2$ if $1 \leq i \leq \mu$.*

Proof. Let the set I be as defined in Step 2 of the algorithm. Then, for distinct i and i' in I we obtain $\alpha(i) \neq \alpha(i')$ using Lemmas 7.9 and 7.10. Thus no multiple edges are created when new edges are added in Step 2, and hence H_2 is a simple graph. The proof of the remaining part of the lemma is obvious. □

The following two lemmas are needed to prove the correctness of Step 3 of Algorithm 4.

LEMMA 7.12. *Suppose $(x_j, y_j) \in H_2$ for some $j > \mu$. Then $(x_j, y_j) \in H_1$. Further, there exists a unique i such that $j \in T(i, j)$, $\alpha(i) < j$ and $i \leq \mu$.*

Proof. Recall that $\theta(i) \neq \alpha(i)$ in Step 2 of Algorithm 4. So no new 'low-degree edges' are created in H_2, i.e., no edge of the form (x_k, y_k), where $k \geq \mu + 1$, is introduced into H_2. Thus $(x_j, y_j) \in H_1$. The rest of the proof is similar to the proof of Lemma 7.9. □

LEMMA 7.13. *Let $i \leq \mu$. There exists at most one j such that $j \in T(i, j)$. In that case, $j > \alpha(i)$ and $j > \mu$.*

Proof. The proof is similar to the proof of Lemma 7.10. □

LEMMA 7.14. *The bipartite multigraph H_3 computed in Step 3 of Algorithm 4 is a simple bipartite graph that realizes (c, c). Moreover, $(x_i, y_i) \in H_3$ if and only if $1 \leq i \leq \mu$.*

Proof. Let J, $\tau(j)$ and $\psi(j)$ be as in Step 3 of the algorithm. For distinct j and j' in J we obtain $\tau(j) \neq \tau(j')$ using Lemmas 7.12 and 7.13. Thus no multiple edges are created when new edges are added in Step 3 and hence H_3 is a simple graph. Clearly, H_3 is a realization of (c, c) and $(x_j, y_j) \notin H_3$ for all $\mu + 1 \leq j \leq n$. By Lemma 7.11, $(x_i, y_i) \in H_2$ for all $1 \leq i \leq \mu$. The fact that $\psi(j) \neq \tau(j)$ implies that no such edge (high-degree edge) of H_2 is destroyed when some edges are deleted in Step 3. □

Proof of Lemma 7.8: The correctness of the algorithm follows from Lemma 7.14. Recall that the graphs H_0, H_1, H_2, H_3 are represented implicitly using the neighborhoods of vertices in Y. The complexity bounds for Step 1 follow from Theorem 6.5.

Suppose the neighborhood of some vertex y_k changes in Step 2. Then one of the following must hold: $k \in I$ or $k = \alpha(i)$ for some $i \in I$. The following claim states that the neighborhood of y_k changes by exactly one vertex.

Claim 1: $|N_2(y_k) - N_1(y_k)| = 1$.

Consider first the case $k \in I$. Lemma 7.9 implies that $\alpha(i) \geq \mu + 1$ for all $i \in I$. Since $k \leq \mu$, $k \neq \alpha(i)$ for any $i \in I$. Consider now the case $k = \alpha(i)$ for some $i \in I$. Then i is unique by Lemmas 7.9 and 7.10, completing the proof of the claim. Since $N_1(y)$ is a union of at most two disjoint intervals of vertices for all $y \in Y$, we can compute all $\theta(i)$'s in constant time using $O(n)$ processors. Further, the graph H_2 in Step 2 can be computed in constant time using $O(n)$ processors.

Suppose the neighborhood of some vertex y_k changes in Step 3. Then one of the following must hold: $k \in J$ or $k = \tau(j)$ for some $j \in J$. The following claim states that the neighborhood of y_k changes by exactly one vertex.

Claim 2: $|N_3(y_k) - N_2(y_k)| = 1$.

Consider first the case $k \in J$. Lemma 7.12 implies that $\tau(j) \leq \mu$ for all $j \in J$. Since $k \geq \mu + 1$, $k \neq \tau(j)$ for any $j \in J$. Consider now the case $k = \tau(j)$ for some $j \in J$. Then j is unique by Lemmas 7.12 and 7.13, completing the proof of the claim. All $\tau(j)$'s can be computed in constant time using $O(n)$ processors. Claim 1 implies that $N_2(y)$ is a union of at most three disjoint intervals of vertices for all $y \in Y$. Hence all $\psi(j)$'s can be computed in constant time using $O(n)$ processors. Further, the graph H_3 in Step 3 can be computed in constant time using $O(n)$ processors. □

Acknowledgments. We thank the referees for their comments and suggestions for improving the presentation of this paper. We also thank Shiva Chaudhuri for useful discussions.

REFERENCES

[1] A. Aho, J. Hopcroft and J. Ullman *Design and Analysis of Computer Algorithms*, (Exercises on pp. 71-72), Addison-Wesley, 1974.
[2] S. R. Arikati, *New Results in Algorithmic Graph Theory*, Ph.D. thesis, Department of Mathematics, University of Illinois, Chicago, 1993.
[3] S. R. Arikati and U. N. Peled, *Degree sequences and majorization*, Linear Algebra Appl., 199 (1994), pp. 179–211.
[4] T. Asano, *Graphical degree sequence problems with connectivity requirements*, Lecture Notes in Computer Science, Springer-Verlag, 762 (1993), pp. 38-47.
[5] B. Awerbuch, A. Israeli and Y. Shiloach, *Finding Euler circuits in logarithmic parallel time*, proceedings of the ACM Symposium on Theory of Computing, ACM Press, 1984, pp. 249-257.
[6] C. Berge, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.
[7] F.T. Boesch ed., *Large-Scale Networks: Theory and Design*, IEEE Press, New York, 1976.
[8] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.
[9] A. Dessmark, A. Lingas and O. Garrido, *On the parallel complexity of maximum f-matching and the degree sequence problem*, Lecture Notes in Computer Science, Springer-Verlag, 841 (1994), pp. 316–325.
[10] P. Erdős and T. Gallai, *Graphs with prescribed degrees of vertices (in Hungarian)*, Mat. Lapok, II (1960), pp. 264–272.
[11] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, New Jersey, 1962.
[12] D. R. Fulkerson, A. J. Hoffman, and M. H. McAndrew, *Some properties of graphs with multiple edges*, Can. J. Math., 17 (1965), pp. 166–177.
[13] D. Gale, *A theorem on flows in networks*, Pacific J. Math., 7 (1957), pp. 1073–1082.
[14] T. Hagerup and C. Rüb, *Optimal merging and sorting on the EREW PRAM*, Infor. Proc. Letters, 33(4) (1989), pp. 181-185.
[15] F. Harary, *Graph Theory*, Addison-Wesley, New York, 1969.
[16] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, New York, 1952.
[17] A. Israeli and Y. Shiloach, *An improved parallel algorithm for maximal matching*, Infor. Proc. Letters, 22(2) (1986), pp. 57–60.
[18] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, New York, 1992.
[19] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier, Amsterdam, Vol. A, 1990, pp. 869–942.
[20] R. M. Karp, E. Upfal, and A. Wigderson, *Constructing a maximum matching in random NC*, Combinatorica, 6(1) (1986), pp. 35–48.
[21] L. Lovász and M. Plummer, *Matching Theory*, Academic Press, Budapest, Hungary, 1986.
[22] A. W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and its Applications*, Academic Press, New York, 1979.
[23] U. N. Peled and M. K. Srinivasan, *The polytope of degree sequences*, Linear Algebra Appl., 114 (1989), pp. 349–377.
[24] H. J. Ryser, *Combinatorial properties of matrices of zeros and ones*, Canad. J. Math., 9 (1957), pp. 371–377.

[25] G. Sierksma and H. Hoogeveen, *Seven criteria for integer sequences being graphic*, Journal of Graph Theory, 15 (1991), pp. 223–231.
[26] R. I. Tyshkevich, A. A. Chernyak and Zh. A. Chernyak, *Graphs and degree sequences I*, Cybernetics, 23 (1987), pp. 734-745.
[27] J. van Leeuwen, *Graph algorithms*, in J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier, Amsterdam, vol. A, 1990, pp. 525–631.