On the Parallel Complexity of Degree Sequence Problems*

Srinivasa R. Arikati
Max-Planck Institute für Informatik
Im Stadtwald, D-66123 Saarbrücken
Germany

arikati@mpi-sb.mpg.de

Classification: Algorithms and data structures, Parallel algorithms.

Abstract

An integer sequence d is called a degree sequence if there exists a simple graph G such that the degrees of its vertices are precisely the components of d; in that case, G is a realization of d. Given d and an integer k, we study two problems: (i) compute a k-edge-connected realization of d, (ii) compute a k-vertex-connected realization of d. The main contributions of this paper are the first parallel algorithms for these problems. Specifically, we show that problem (i) can be solved in $\tilde{O}(k)$ time using a polynomial number of processors. For problem (ii) we present an efficient algorithm when $k = 2$; the algorithm runs in logarithmic time using a linear number of processors.

1 Introduction

1.1 Problem Definition

A fundamental problem in graph algorithms is to compute a (simple) graph satisfying the given degree constraints and having good connectivity properties. More formally, an integer sequence d is called a *degree sequence* if there exists a graph G such that the degrees of its vertices are precisely the components of d; in that case G is said to be a *realization* of d.

Given d and an integer k, we study the parallel complexity of following problems.

Problem 1: Compute a k-edge-connected realization of d.
Problem 2: Compute a k-vertex-connected realization of d.

Degree sequence problems have several applications in diverse areas such as network reliability, structural reliability and stereochemistry (see [18, 21]).

1.2 Previous Results

On the distinction between search and decision problems in parallel computation, Upfal, Karp and Wigderson said [19], "In the context of parallel computation the distinction between search

*This work is partially supported by the EEC ESPRIT Basic Research Action No. 7141 (ALCOM II).

problems and decision problems is far more important [than in sequential computation] because we are interested in algorithms that run in sublinear time. And in fact there are many cases where a decision problem is easy or even trivial to solve in parallel, but the corresponding search problem is challenging." Degree sequence problems are one such important case. The decision problems can be solved by verifying certain linear inequalities and the verification can be done easily and efficiently in parallel (cf. Section 2). The status of the search problems, on the other hand, has been open so far.

Recently several authors worked on the problem of computing any realization (no connectivity requirements). Dessmark, Lingas, and Garrido [5] showed that a special case of this problem is in NC. Arikati and Maheshwari [1, 2] showed that the general problem has an efficient deterministic parallel solution; their algorithm runs in logarithmic time using a linear number of CRCW PRAM processors.

1.3 New Results

The main contributions of this paper are the first parallel algorithms for the problems defined above. Specifically, we obtain the following results, where n and m denote the number of vertices and edges in the realization.

- A randomized parallel algorithm for Problem 1 that runs in $\tilde{O}(k)$ time[1] using $O(n^{4.5}m)$ CRCW PRAM processors; a deterministic parallel algorithm for the same problem that runs in $\tilde{O}(k)$ time using a polynomial number of processors (the polynomial has a very high degree).
- An efficient deterministic parallel algorithm for Problem 2 when $k = 2$; the algorithm runs in $O(\log n)$ time using $O(n + m)$ CRCW PRAM processors.

1.4 An Overview

Our basic technique is to start with any realization of d, make appropriate modifications on G without leaving the space of all realizations, and finally reach a realization that has the desired properties. The technique is based on a natural, fundamental operation called exchange (see Figure 2 in Section 2) that transforms one realization into another.

The algorithm to compute a connected (i.e. $k = 1$) realization of d is simple and efficient. Starting with any realization G, we compute connected components of G and merge these components by performing appropriate exchanges. We show that many exchanges can be done in parallel. The algorithm runs in deterministic $O(\log n)$ time using $O(n+m)$ PRAM processors. The details are presented in Section 3.

Wang and Kleitman [20] presented in 1973 an algorithm to solve Problem 2, and Asano [3] recently proposed an efficient sequential implementation of their algorithm. We partially solve Problem 2 by presenting a deterministic NC algorithm when $k = 2$. Using the results of Section 3 we compute a connected realization G of d. We exploit the structure of the block-cutvertex tree of G to compute a 2-vertex-connected realization efficiently. The algorithm is presented in Section 4.

Edmonds [7] presented in 1964 an algorithm to solve Problem 1. Our algorithm is based on a non-trivial parallelization of his algorithm. We use an incremental approach to compute

[1] $\tilde{O}(k)$ means $O(k \text{ polylog}(n))$.

a k-edge-connected realization of d, namely we show how to transform an i-edge-connected realization into an $(i+1)$-edge-connected realization. We introduce the notion of extreme sets and show that many extreme sets can be destroyed in one step by performing exchanges in parallel. The extreme sets can be computed from the elegant cactus representation [6, 16], which compactly represents all connectivity cuts of a graph. We give in Section 5 a brief description of this representation. Karzanov and Timofeev [13] presented an efficient sequential algorithm to compute this representation. Naor and Vazirani [17] presented an RNC algorithm to compute the cactus representation. The deterministic NC algorithm of Karger and Motwani [12] to compute min-cuts can be modified to compute this representation [11]. Our algorithm to solve Problem 1 is presented in Section 6.

2 Preliminaries

By a *graph* G we mean a simple graph. We use $V(G)$ and $E(G)$ to denote, respectively, the vertex set and edge set of G. If uv is an edge in G, we say that $uv \in G$.

Throughout this paper, V ($|V| = n$) denotes a fixed set that usually stands for the vertex set of the graph under consideration. Also, $d = (d_1, \ldots, d_n)$ denotes a nonnegative integer sequence, where $d_1 \geq d_2 \geq \cdots \geq d_n$; we define $m = (1/2)\sum_{i=1}^{n} d_i$.

The sequence d is *realizable* if there exists a graph G in which the degrees of the vertices are precisely the components of d; in that case G *realizes* d (see Figure 1). The sequence d is a *connected degree sequence* if there exists a connected graph that realizes d; d is a *k-edge-connected degree sequence* (resp. *k-vertex-connected degree sequence*) if there exists a k-edge-connected graph (resp. k-vertex-connected graph) that realizes d.

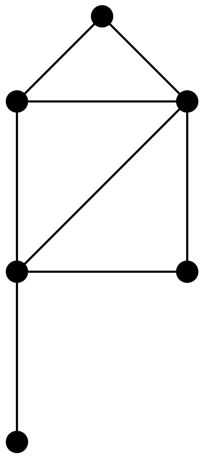

Figure 1: A realization of $(4, 4, 3, 2, 2, 1)$.

Theorem 2.1 *([4, 8, 15]) The sequence d is realizable if and only if (i) $\sum_{i=1}^{n} d_i$ is even and (ii) $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$, for $k = 1, 2, \ldots, n$.*

Theorem 2.2 *([4, 15]) The sequence d is a connected sequence if and only if (i) d is realizable and (ii) $\sum_{i=1}^{n} d_i \geq 2(n-1)$.*

Theorem 2.3 *([7]) The sequence d is a k-edge-connected sequence ($k \geq 2$) if and only if (i) d is realizable and (ii) $d_i \geq k$ for all i.*

Theorem 2.4 *([20]) The sequence d is a k-vertex-connected sequence ($k \geq 2$) if and only if (i) d is realizable, (ii) $d_i \geq k$ for all i, and*

$$(iii)\ m - \sum_{i=1}^{k-1} d_i + \frac{(k-1)(k-2)}{2} \geq n - k.$$

Given d, a decision problem is to test if, for example, d is k-edge-connected, and the search problem is to actually compute a k-edge-connected realization of d. It is clear that the decision problems associated with the above four theorems are all in NC: After sorting d, all the inequalities given in these theorems can be checked in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. When we discuss a search problem associated with d we can therefore assume that the corresponding decision problem is solvable.

We use a fundamental operation to transform one realization of d into another. Suppose G is a realization of d, and let u, v, w, x be four vertices such that $uv, wx \in G$ and $uw, vx \notin G$. We say that (u, v, w, x) is an *exchange sequence.* An *exchange* on (u, v, w, x) consists of dropping the edges uv, wx and adding the edges uw, vx (see Figure 2).

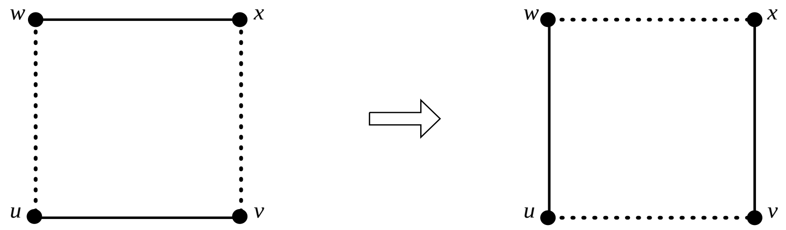

Figure 2: An exchange operation. A solid line indicates the presence of an edge and a dotted line its absence.

An important observation is the following:

Lemma 2.1 *If G' is obtained from G by performing an exchange on (u, v, w, x), then G' is also a realization of d.*

A *bridge* (resp. *cut-vertex*) in a connected graph G is an edge (resp. a vertex) whose removal disconnects G.

3 Connected Degree Sequences

Throughout this section, we assume $d = (d_1, \ldots, d_n)$ is a connected degree sequence. For convenience, we reproduce the following condition from Theorem 2.2:

$$\sum_{i=1}^{n} d_i \geq 2(n-1). \tag{1}$$

Let $G = (V, E)$ be any realization of d. Assuming that G is not connected, we present an algorithm that transforms G into a connected realization G^*.

A connected component of G is called *big* if it contains a cycle; otherwise it is *small*. Since d satisfies Equation 1 there must be at least one big component. We begin with the following simple lemma.

Lemma 3.1 *1. If C is a small component, then $|E(C)| = |V(C)| - 1$.*

2. If T is any spanning tree in a big component C, then the number of non-tree-edges in C is $|E(C)| - |V(C)| + 1$.

Our parallel algorithm is based on the following natural operation to merge components of G.

Lemma 3.2 *Suppose C is a big component and let uv be any non-bridge in it. Let $C^{'}$ be any other (big or small) component and wx be any edge in it. If $\hat{G}$ is obtained by performing an exchange on (u, v, w, x), then $\hat{G}$ is a realization of d and has fewer components than G.*

Proof: $C \cup C^{'} - \{uv, wx\} \cup \{uw, vx\}$ is a component in $\hat{G}$. ∎

The algorithm is given in Algorithm 1.

1. Compute a realization G of d.
2. Compute connected components of G. For all components C, find a spanning tree of C; NT_C is the set of all the non-tree edges of C.
3. **while** there are more than one big component **do**
   (a) Group the big components into pairs.
   (b) For all pairs $(C, C^{'})$ do in parallel: pick $uv \in NT_C$ and $wx \in NT_{C^{'}}$ and perform an exchange on (u, v, w, x). Set $NT_{\hat{C}} := NT_C \cup NT_{C^{'}} - \{uv, wx\} \cup \{uw\}$, where $\hat{C} = C \cup C^{'} - \{uv, wx\} \cup \{uw, vx\}$.
4. Suppose the resulting graph $G^{'}$ is still disconnected. Let $C^{'}$ be the big component, and $C_1, \ldots, C_p$ be the small components. Let $(u_1, v_1), \ldots, (u_q, v_q)$ be the edges in $NT_{C^{'}}$. (*Comment*: The claim below shows that $q \geq p$.)
   Pick an edge, say (w_i, x_i), from C_i and perform simultaneous exchanges on (u_i, v_i, w_i, x_i) for $i = 1, \ldots, p$.
5. Output the resulting graph G^{*}.

Algorithm 1: Computing a connected realization.

Theorem 3.1 *Algorithm 1 computes a connected realization of a connected degree sequence $d = (d_1, \ldots, d_n)$ in $O(\log n)$ time using $O(n + m)$ CRCW PRAM processors.*

Proof: Observe that all the edges in NT_C are non-bridges in the component C. In Step 3(b), out of the two possible edges uw and vx, we include uw in $NT_{\hat{C}}$ (vx merges C and C' into one component). Repeated applications of Lemma 3.2 shows that G' is a realization of d. If G' is disconnected, it has exactly one big component. The following claim shows that p simultaneous exchanges can be performed in Step 4.

Claim. $q \geq p$.

Let n' and m' denote the number of vertices and edges in C', and n_i and m_i denote the number vertices and edges in C_i. Lemma 3.1 implies $m_i = n_i - 1$ and $|NT_{C'}| = m' - n' + 1$. Since d satisfies Equation 1, the total number of edges $m' + \sum_{i=1}^{p} m_i$ is at least $n - 1$. The claim follows by observing that $n' + \sum_{i=1}^{p} n_i$ equals n, the total number of vertices. Repeated application of Lemma 3.2 again shows that G^* is a connected realization of d.

The algorithm of [1] computes G in $O(\log n)$ time using $O(n+m)$ CRCW PRAM processors. The while loop in Step 3 is iterated for at most $O(\log n)$ times and each iteration takes constant time using $O(m)$ processors. The remaining steps can be done within these resource bounds using standard PRAM algorithms [10]. ∎

4 2-Vertex-Connected Degree Sequences

Throughout this section, d denotes a 2-vertex-connected degree sequence. With $k = 2$, condition (ii) of Theorem 2.4 becomes $d_i \geq 2$ for all i, and condition (iii) is

$$m - d_1 \geq n - 2. \tag{2}$$

Let G be any realization of d. Based on the results of Section 3 we may assume that G is connected. A *block* (also called a 2-connected component) of G is a maximal 2-vertex-connected subgraph of G. Two blocks are *adjacent* if they share a vertex. (It is well known that any two blocks can have at most one common vertex and that the common vertex is a cutvertex in G.) The nodes of the so-called *block-cutvertex tree* are the blocks and cutvertices of G, and the edges of T are of the form (B, v) where B is a block and $v \in B$ is a cutvertex. A block B'' is said to be on the B–B' path if the B–B' path in T passes through B''.

For $X \subseteq V$, $G[X]$ denotes the subgraph of G induced by X. The union of two graphs H and H' is denoted by $H \cup H'$; H and H' may have common vertices, but they will always be edge-disjoint in this paper.

The intuition behind obtaining a 2-connected realization G^* of d is to merge all blocks of G into a single block by performing appropriate exchanges. The following two lemmas form the basis of our algorithm. Due to space considerations, the proofs of the lemmas are left to the final version of the paper.

Lemma 4.1 *Let B and B' be adjacent blocks with common vertex z. Suppose $B - z$ contains a cycle and let uv be a non-bridge in $B - z$. Let wx be an edge in B' such that (u, v, w, x) is an exchange sequence. Finally, let $\hat{G}$ be obtained by performing an exchange on (u, v, w, x). Then:*

1. *$\hat{G}$ is connected.*

2. *$\hat{G}[V(B) \cup V(B')]$ is a block in $\hat{G}$.*

Lemma 4.2 *Let B and $B^{'}$ be nonadjacent blocks, and let (u, v, w, x) be an exchange sequence where $u, v \in B$ and $w, x \in B^{'}$. Define C to be the union of all blocks (including B and $B^{'}$) on the B–$B^{'}$ path. Let $\hat{G}$ be obtained by performing an exchange on (u, v, w, x). Then:*

1. *$\hat{G}$ is connected.*
2. *$\hat{G}[V(C)]$ is a block in $\hat{G}$.*

Our algorithm to compute the desired realization G^* consists of two phases. In the first phase we compute a realization $G^{'}$ that has at most one cutvertex. If G has exactly two blocks, we take $G^{'} = G$ and go to the second phase.

Phase 1. Compute the block-cutvertex tree T, root it at any non-leaf, and label its leaves from left to right as $B_1, B_2, \ldots, B_p$. Group the leaves into pairs (B_i, B_{i+q}), $i = 1, \ldots, q$, where $q = p/2$. A pair (B, B') is called *active* if B and $B^{'}$ are nonadjacent. For all active pairs (B, B') do in parallel: select $uv \in B$ and $wx \in B^{'}$, and perform an exchange on (u, v, w, x). Denote the resulting graph by $G^{'}$.

Lemma 4.3 *$G^{'}$ realizes d and has at most one cutvertex.*

Proof: First assume that p is even. Every block B_j has at least three vertices, since B_j is a leaf and $d_i \geq 2$ for all i. Set $G_0 = G$ and let G_i be obtained from G_{i-1} by performing an exchange on (u, v, w, x), where (u, v, w, x) is as chosen by the algorithm for the active pair (B_i, B_{i+q}). Lemma 4.2 shows that $G_i[V(C)]$ is a block in G_i, where C is the union of all blocks on the B_i–B_{i+q} path in the block-cutvertex tree of G_{i-1}. Because T is rooted at a non-leaf, it follows that $G^{'} = G_q$ has at most one cutvertex.

If p is odd, we take $q = \lfloor p/2 \rfloor$ and group the leaves of T into pairs (B_i, B_{i+q+1}) and then perform the exchanges as above. ∎

Phase 2. Let z be the cutvertex of in $G^{'}$. Define $H = G^{'} - z$ and let $c = (c_1, \ldots, c_{n-1})$ be the degree sequence of H. Using Algorithm 1 compute a connected realization H^* of c. Then $G^* = H^* \cup \{z\}$, where $uz \in G^*$ iff $uz \in G^{'}$.

Lemma 4.4 *G^* is a 2-vertex-connected realization of d.*

Proof: We claim that c is a connected degree sequence. By definition, c is realizable. Since d satisfies Equation 2, it follows that c satisfies Condition (ii) of Theorem 2.3. This proves the claim. Recall that in Algorithm 1 big components are merged first and then the small components are all merged with the remaining big component. This fact combined with Lemma 4.1 proves that G^* is a 2-vertex-connected realization of d. ∎

We summarize the main result.

Theorem 4.1 *A 2-vertex-connected realization of d can be computed in $O(\log n)$ (deterministic) time using $O(n + m)$ CRCW PRAM processors.*

Proof: The complexity of our algorithm is dominated by two factors: the construction of the block-cutvertex tree and the computation of H^*. Using standard PRAM algorithms [10], the former can be done within the resource bounds stated in the theorem; the complexity of the latter follows from Theorem 3.1. ∎

5 Cactus Representation

Given a connected graph $G = (V, E)$, a *cut* $(X, \overline{X})$ is a partition of the vertices into two nonempty sets X and $\overline{X}$. An edge uv *belongs* to the cut $(X, \overline{X})$ if one of u and v is in X and the other in $\overline{X}$. The *value* of a cut is the number of edges that belong to it. The *edge-connectivity* of G, denoted by $\lambda(G)$, is the minimum number of edges whose removal disconnects G, or equivalently, $\lambda(G)$ is defined to be the minimum value of a cut. If there are weights on the edges, then the value of a cut is the total weight of edges that belong to it. A cut whose value equals $\lambda(G)$ is called a *connectivity cut.*

A graph G is called a *cactus graph* if any two cycles can have at most one vertex in common. The edges of a cactus graph can be partitioned into *cycle-edges* (edges that lie on cycles) and *tree-edges.*

Let $G = (V, E)$ be a cactus graph with edge-weights $w(e)$ as follows: $w(e) = \frac{\lambda}{2}$ if e is a cycle-edge, and $w(e) = \lambda$ if e is a tree-edge. Let u and v be two vertices in G. By merging u and v, we obtain a new graph $G' = (V', E')$ with edge-weights $w'(e)$ as follows. $V' = V - v$; $E' = E - \{vw : vw \in E\} \cup \{uw : w \neq u \text{ and } vw \in E\}$; $w'(uw) = w(uw) + w(vw)$, and $w'(e) = w(e)$ for all other edges e. It is easy to see that G' is also a cactus, and that $w'(e) = \frac{\lambda}{2}$ if e is a cycle-edge, and $w'(e) = \lambda$ if e is a tree-edge.

Dinits, Karzanov and Lomosonov [6] derived the compact and elegant *cactus representation* $\mathcal{H} = \mathcal{H}(G)$ of a graph $G = (V, E)$. We give a brief description of $\mathcal{H}$; more details can be found in [13, 16, 17]. $\mathcal{H}$ is an edge-weighted cactus graph of $O(n)$ nodes and edges. Every vertex in G maps to exactly one node in $\mathcal{H}$ and any node in $\mathcal{H}$ corresponds to a subset (possibly empty) of vertices from G. A cut $(S, \overline{S})$ in $\mathcal{H}$ induces a cut $(X, \overline{X})$ in G, where X consists of all vertices from G that are mapped into nodes in S. The edge-connectivity of $\mathcal{H}$ is also $\lambda(G)$. Each connectivity cut in $\mathcal{H}$ induces a connectivity cut in G, and each connectivity cut in G corresponds to one or more connectivity cuts in $\mathcal{H}$. Every cycle-edge of $\mathcal{H}$ is given a weight $\frac{\lambda}{2}$ and every tree-edge is given a weight λ. The connectivity cuts in $\mathcal{H}$ are of exactly two types: (i) a cut obtained by removing a tree-edge, and (ii) a cut obtained by removing any pair of cycle-edges that lie on the same cycle.

6 k-Edge-Connected Degree Sequences

In this section d is a k-edge-connected degree sequence ($k \geq 2$). Hence $d_i \geq k$ for all i.

Starting with any realization $G = (V, E)$ of d, we show how to compute a k-edge-connected realization. Based on the results from Section 3, we may assume that G is connected. We use an incremental approach. Assume inductively that G is $(k-1)$-edge connected. We may assume that G is not k-edge connected.

For $U \subset V$, let $d(U)$ denote the 'degree' of U, i.e., $d(U) = |\{uv : u \in U, v \in \overline{U}\}|$, where $\overline{U} = V - U$. U is a *critical set* if $d(U) \leq k - 1$ (in fact $d(U) = k - 1$ because G is $(k-1)$-edge connected); U is an *extreme set* if (1) U is critical and (2) no proper subset of U is critical. Observe that singleton sets are not extreme (as $d_i \geq k$) and that there is at least one extreme set (as G is not k-edge connected). A straightforward observation is that any two extreme sets must be disjoint, i.e., $W \subseteq \overline{U}$ for all extreme sets U and W.

The intuition behind obtaining the desired realization of d is to destroy all critical sets without creating any new ones. Suppose U is an extreme set in G and let ux be an edge such that $u \in U$ and $x \in \overline{U}$. Since $d_u \geq k$ and $d(U) \leq k-1$, there exists a vertex $v \in U$ such that $uv \in G$, $vx \notin G$. Similarly there exists a vertex $w \in \overline{U}$ such that $wx \in G$, $uw \notin G$, since $d_x \geq k$ and $d(\overline{U}) = d(U) \leq k-1$. Thus (u, v, w, x) is an exchange sequence. Let $\hat{G}$ be obtained from G by performing an exchange on (u, v, w, x).

Lemma 6.1 *([7])*

1. *$\hat{G}$ is a realization of d and U is not critical in $\hat{G}$.*

2. *Every critical set in $\hat{G}$ is also a critical set in G.*

Proof: Proof of (1) is straightforward. Denote the degree of X in $\hat{G}$ by $\hat{d}(X)$. To get a contradiction, assume that $(X, \overline{X})$ is critical in $\hat{G}$ but not in G. So $\hat{d}(X) \leq k-1$ and $d(X) \geq k$. It follows that both u, w are in one side of the cut $(X, \overline{X})$ and both v, x are in the other side; say $u, w \in X$ and $v, x \in \overline{X}$. Put $p = \hat{d}(U \cap X, U \cap \overline{X})$, $q = \hat{d}(\overline{U} \cap X, \overline{U} \cap \overline{X})$, $r = \hat{d}(U \cap X, \overline{U} \cap \overline{X})$, where $\hat{d}(P, Q)$ denotes the number of edges in $\hat{G}$ with one endvertex in P and the other in Q. Then $r \geq 1$ since $ux \in \hat{G}$, $u \in U \cap X$, and $x \in \overline{U} \cap \overline{X}$. This fact combined with $\hat{d}(X) = p + q + r$ implies $p + q \leq k - 2$. We consider the case $p \leq \lfloor (k-2)/2 \rfloor$; the proof for the other case, namely $q \leq \lfloor (k-2)/2 \rfloor$, is similar. The following claim implies that U is not extreme in G, a contradiction.

Claim. $d(U \cap X) \leq k - 1$ or $d(U \cap \overline{X}) \leq k - 1$.

Put $s = |(U \cap X, \overline{U})|$ and $t = |(U \cap \overline{X}, \overline{U})|$, where $|(P, Q)|$ denotes the number of edges in G with one endvertex in P and the other in Q. Then $k - 1 \geq d(U) = s + t$, implying that either (i) $s \leq \lfloor (k-1)/2 \rfloor$ or (ii) $t \leq \lfloor (k-1)/2 \rfloor$. If (i) holds, then $d(U \cap X) = p + 1 + s \leq k - 1$ because $d(U \cap X) = |(U \cap X, U \cap \overline{X})| + |(U \cap X, \overline{U})|$, and $|(U \cap X, U \cap \overline{X})| = p + 1$; the second equality follows from the fact that $uv \notin \hat{G}$, $uv \in G$. Similarly we can prove $d(U \cap \overline{X}) \leq k - 1$ if (ii) holds. ∎

6.1 The Algorithm

Our algorithm to compute a k-edge-connected realization of d consists of several phases, and each phase has 4 steps.

Let $\mathcal{S}$ be the set of extreme sets in G. Define the *extreme sets graph* $\mathcal{G}$ as follows. $V(\mathcal{G}) = \mathcal{S}$ and two vertices U and W of $\mathcal{G}$ are adjacent iff they are 'adjacent' in G, i.e., iff there is an edge in G that joins a vertex in U to a vertex in W.

Step 1. Let M be a maximal matching in $\mathcal{G}$. For all edges $(U, W) \in M$ do in parallel: Let ux be an edge in G such that $u \in U$ and $x \in W$; select vertices v and w such that $v \in U$, $w \in W$, and (u, v, w, x) is an exchange sequence; perform an exchange on (u, v, w, x).

The resulting graph is denoted by G'. Let $\mathcal{S}' = \{W \in \mathcal{S} : W \text{ is extreme in } G'\}$.

Lemma 6.2 *G' realizes d. Further, no two extreme sets of $\mathcal{S}'$ are adjacent.*

Proof: Repeated applications of Lemma 6.1 shows that G' realizes d. The second part of the lemma follows from the fact that M is a maximal matching in $\mathcal{G}$. ∎

We discuss Step 2 now. Suppose $U \in \mathcal{S}'$. Let vertex v be the smallest neighbor of U, i.e., (i) $v \in \overline{U}$, (ii) there is an edge that joins v to a vertex in U, and (iii) v is the smallest such vertex[2]. Observe that v exists. The *parent* of U is defined as v and U is a *child* of v. A vertex is called *active* if it has a child. An active vertex is called *big* if it has at least two children.

Lemma 6.3 *Let $U \in \mathcal{S}'$ and let v be the parent of U. Then there exist vertices $u, x \in U$ such that $uv, ux \in G'$ and $vx \notin G'$.*

Proof: Since v is the parent of U there exists $u \in U$ such that $uv \in G'$. The existence of the desired vertex x follows from the facts $d(U) \leq k-1$ and $d_u \geq k$. ∎

Step 2. For all big vertices v do in parallel: Let $U_1, \ldots, U_p$ be the children of v, and u_i be any vertex in U_i such that $vu_i \in G'$. Let x_i be any vertex in U_i such that $u_i x_i \in G'$ and $vx_i \notin G'$. Drop the edges $\{u_i x_i : 1 \leq i \leq p\}$, $\{vu_i : 1 \leq i \leq p-2\}$, and vu_p; add the edges $\{vx_i : 1 \leq i \leq p-1\}$, $u_1 x_p$, $u_1 u_p$, $u_2 u_p$, $\{u_i u_{i+1} : 2 \leq i \leq p-2\}$ (see Figure 3). Let G'' be the resulting graph.

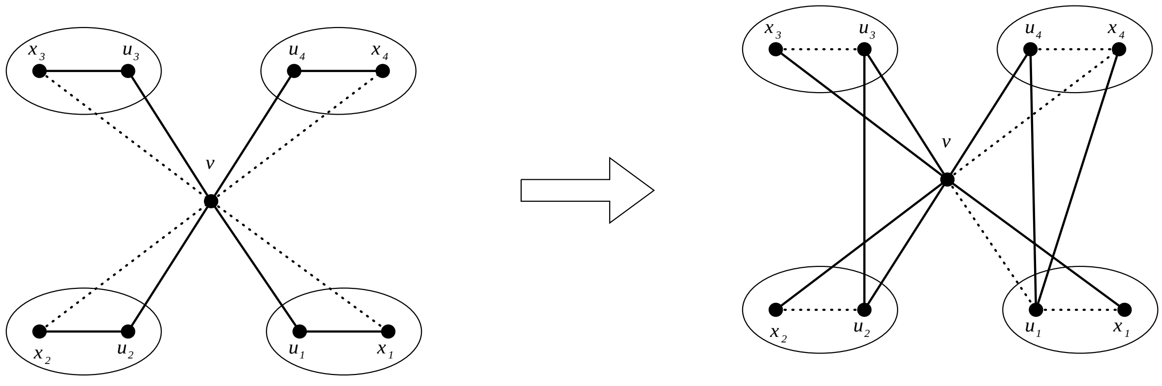

Figure 3: Step 2 of the algorithm. A solid line indicates the presence of an edge and a dotted line its absence.

Lemma 6.4 *G'' realizes d.*

Proof: Fix a big vertex v. Lemma 6.2 shows that the following are all exchange sequences: (i) (u_1, x_1, u_p, v), (ii) (u_i, x_i, u_{i-1}, v) for $2 \leq i \leq p-2$ and (iii) (u_p, x_p, u_2, u_1). Let $G_0 = G', G_1, \ldots, G_p$ a sequence of graphs such that G_1 is obtained from G_0 be performing an exchange on (u_1, x_1, u_p, v), G_i $(2 \leq i \leq p-2)$ is obtained from G_{i-1} by performing an exchange

[2]Here and later, any fixed v suffices; for the sake of definiteness, we choose the smallest v according to a fixed linear ordering of the vertices.

on (u_i, x_i, u_{i-1}, v), and G_p is obtained from G_{p-1} by performing an exchange on (u_p, x_p, u_2, u_1). Lemma 6.1 proves that G_p is a realization of d. Now, Lemma 6.2 implies that the parent of an extreme set doesn't belong to any extreme set. So no multiple edges are created when the exchanges are performed in parallel for all big vertices. ∎

Let $\mathcal{S}'' = \{W \in \mathcal{S}' : W \text{ is extreme in } G''\}$. Let v be an active vertex in G''. Since v is not big, it has a unique child $U \in \mathcal{S}''$. Let $f(v)$ be the smallest neighbor of v in U, and $g(v)$ be the smallest $u \in U$ such that $(f(v), u) \in G'$ and $vu \notin G'$. The *friend* of v, denoted by $h(v)$, is defined as the smallest vertex in $\{w : w \notin U, vw \in G', (f(v), w) \notin G'\}$. Define a new graph H as follows. The vertices of H are the active vertices in G', and two vertices v and w are adjacent in H iff they are mutual friends (i.e., $h(v) = w$ and $h(w) = v$).

Observe that every vertex in H has degree either 0 or 1.

Step 3. For all edges vw in H, do the following step in parallel on the graph G'': drop the edges $(f(v), g(v))$, vw, $(f(w), g(w))$, and add the edges $(v, g(v))$, $(f(v), f(w))$, $(w, g(w))$.

It is routine to prove that the resulting graph, denoted by $G^{(3)}$, realizes d. Let $\mathcal{S}^{(3)} = \{W \in \mathcal{S}'' : W \text{ is extreme in } G^{(3)}\}$.

Step 4. For all active vertices v in $G^{(3)}$ such that the child of v is in $\mathcal{S}^{(3)}$, do the following step in parallel on the graph $G^{(3)}$: Perform exchange on $(f(v), g(v), h(v), v)$.

The resulting graph is denoted by $G^{(4)}$. This completes the description of one phase of the algorithm. The proof of the following lemma follows from above discussion.

Lemma 6.5 *$G^{(4)}$ realizes d.*

6.2 The Correctness and Complexity

Our proof of correctness and the complexity analysis of the algorithm is based on properties and manipulations of cactus representation.

Let $\mathcal{H} = \mathcal{H}(G)$ be a cactus representation of G. A node W is a *leaf* in $\mathcal{H}$ if W is either connected by a single tree-edge, or W is in a cycle and has degree exactly two. By the definition of $\mathcal{H}$ it follows that extreme sets in G are precisely the leaves in $\mathcal{H}$. We denote the set of leaves of $\mathcal{H}(G)$ by $L(G)$.

Recall that G is a realization of d and that $\lambda(G) = k - 1$. Let U and W be two adjacent extreme sets in G. Select an edge ux G such that $u \in U$ and $x \in W$; select vertices v and w such that $v \in U$, $w \in W$, (u, v, w, x) is an exchange sequence. Obtain another realization $\hat{G}$ from G by performing an exchange on (u, v, w, x). Call a node Y of $\mathcal{H}$ *active* if either $Y \in \{U, W\}$ or (Y is on a U-W path in $\mathcal{H}$ and Y doesn't lie on a cycle).

Lemma 6.6 *Let S be the set of active nodes. If $S = V(\mathcal{H})$, then $\lambda(\hat{G}) \geq k$. Otherwise,*

1. *$\lambda(\hat{G}) = k - 1$;*

2. *The graph $\hat{\mathcal{H}}$ obtained from $\mathcal{H}$ by merging nodes of S is a cactus representation of $\hat{G}$.*

Proof (sketch): Consider first the case $S = V(\mathcal{H})$. Then $\mathcal{H}$ is a path with end-nodes U and W. So every connectivity cut in G separates U and W. Assume if possible that $(X, \overline{X})$ is a cut of value at most $k - 1$ in $\hat{G}$. Since X is critical in $\hat{G}$ it is also critical in G by Lemma 6.1, and the cut $(X, \overline{X})$ has value $k-1$ in G. Since two new edges are added between U and W, the value of this cut in $\hat{G}$ is $2+k-1=k+1$, a contradiction. Hence $\lambda(\hat{G}) \geq k$.

Consider now the case $S \neq V(\mathcal{H})$. Then there exists a connectivity cut in G that doesn't separate the nodes in S. This cut is also a cut in $\hat{G}$ and has the same value ($= k-1$) as in G. Hence $\lambda(\hat{G}) \leq k-1$. It follows from Lemma 6.1 that a connectivity cut in $\hat{G}$ is also a connectivity cut in G and the cut contains both U and W in one side of the cut. The cuts in G that separate U and W are precisely those cuts that are obtained from $\mathcal{H}$ by deleting tree-edges incident on the active nodes. Hence $\hat{\mathcal{H}}$ a cactus representation of $\hat{G}$. ∎

Recall that G' is the graph obtained from G in Step 1 of the algorithm. Repeated applications of Lemma 6.6 show that $|L(G')| \leq |L(G)| - \frac{p}{2}$, where p is the number of leaves of $\mathcal{H}(G)$ that are matched by the matching M in Step 1. For Steps 2–4, similar results can be proved; due to space considerations, these results are left to the final version.

Lemma 6.7 *Let the graph $G^{(4)}$ be defined as above. If $G^{(4)}$ is not k-edge-connected, then $|L(G^{(4)})| \leq \frac{1}{2}|L(G)|$.*

Below we summarize the main result of this section.

Theorem 6.1 *A k-edge-connected realization of d can be computed in $O(k \log^4 n)$ using $O(n^{4.5}m)$ CRCW processors on a probabilistic PRAM.*

Proof (sketch): In each phase of the algorithm, the number of leaves of the cactus reduces by a factor of at least 2, by Lemma 6.7. Hence there are $O(\log n)$ phases. The correctness of each phase follows from Lemma 6.5. The time and processor complexity of a phase is dominated by two subproblems: finding the cactus representation and finding a maximal matching. Naor and Vazirani [17] presented an RNC algorithm to compute the cactus representation and their algorithm runs in $O(\log^2 n)$ using $O(n^{4.5}m)$ CRCW processors. The algorithm of Israeli and Shiloach [9] computes a maximal matching in deterministic $O(\log^3 n)$ time using $O(m)$ CRCW PRAM processors. ∎

Observe from Theorem 6.1 that the search problem for k-edge-connected degree sequences is in RNC for $k = polylog(n)$. The results of Karger and Motwani [12] imply an NC algorithm (though not practical) to compute the cactus representation [11]. Therefore, we have the following.

Theorem 6.2 *The problem of computing a k-edge-connected realization of d can be solved in deterministic $\tilde{O}(k)$ time using a polynomial number of processors.*

7 Conclusions

We presented the first parallel algorithms to solve the degree sequence problems with connectivity requirements. An important open problem is to solve the vertex-connectivity case completely. Our techniques for solving k-vertex-connectivity case (when $k=2$) may not generalize for arbitrary values of k, especially because finding k-blocks is P-complete for all $k \geq 3$ (see [14]).

Acknowledgements: Thanks to Ramesh Hariharan for his helpful comments and to Professor Kurt Mehlhorn for his support and encouragement.

References

[1] S.R. Arikati and A. Maheshwari. Realizing degree sequences in parallel. *Proc. 5th ISAAC'94*, Lecture Notes in Comp. Sci. (LNCS) 834:261–269, Springer-Verlag, 1994. To appear in *SIAM Journal Discrete Math.*

[2] S.R. Arikati and A. Maheshwari. An $O(n)$ algorithm for realizing degree sequences. *Proc. 14th FSTTCS, India*, Lecture Notes in Comp. Sci.(LNCS) 880:125–136, Springer-Verlag, 1994.

[3] T. Asano. Graphical degree sequence problems with connectivity requirements. *Proc. 4th ISAAC'93*, Lecture Notes in Comp. Sci. (LNCS) 762:38–47, Springer-Verlag, 1993.

[4] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.

[5] A. Dessmark, A. Lingas and O. Garrido. On the parallel complexity of maximum f-matching and the degree sequence problem. *Proc. MFCS'94*, Lecture Notes in Comp. Sci. (LNCS) 841: 316–325, Springer-Verlag, 1994.

[6] E.A. Dinits, A.V. Karzanov and M.V. Lomosonov. On the structure of a family of minimal weighted cuts in a graph. In *Studies in Discrete Optimization* [in Russian], (A.A. Fridman, Ed.) pp. 290–306, Nauka, Moscow, 1976.

[7] J. Edmonds. Existence of k-edge connected ordinary graphs with prescribed degrees. *J. Res. Nat. Bur. Standards*, 68(B):73–74, 1964.

[8] F. Harary. *Graph Theory.* Addison-Wesley, New York, 1969.

[9] A. Israeli and Y. Shiloach. An improved parallel algorithm for maximal matching. *Inf. Proc. Letters*, 22(2):57–60, 1986.

[10] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, New York, 1992.

[11] D.R. Karger. Personal Communication, June, 1994.

[12] D.R. Karger and R. Motwani. Derandomization through approximation: An NC algorithm for minimum cuts. Proc. 25th ACM STOC, pp. 497–506, ACM Press, 1993.

[13] A.V. Karzanov and E.A. Timofeev. Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. *Cybernetics*, pp. 156–162, Translated from Kibernetica, No. 2, pp. 8–12, March-April, 1986.

[14] L. Kirousis, M. Serna and P. Spirakis. The parallel complexity of the subgraph connectivity problem. Proc. 29th FOCS, pp. 294–299, IEEE Press, 1989.

[15] L. Lovász and M. Plummer. *Matching Theory.* Academic Press, Budapest, Hungary, 1986.

[16] D. Naor, D. Gusfield and C. Martel. A fast algorithm for optimally increasing the edge connectivity. Proc. 31st FOCS, pp. 698–707, IEEE Press, 1991.

[17] D. Naor and V.V. Vazirani. Representing and enumerating edge connectivity cuts in RNC. *Proc. 2nd WADS*, Lecture Notes in Comp. Sci. (LNCS) 519:273–285, Springer-Verlag, 1991.

[18] R.I. Tyshkevich, A.A. Chernyak and Zh. A. Chernyak. Graphs and degree sequences I. *Cybernetics*, 23:734-745, 1987.

[19] E. Upfal, R.M. Karp and A. Wigderson. The complexity of parallel search. IBM Research Report, RJ 5434 (55563), N.Y., 1986.

[20] D.L. Wang and D.J. Kleitman. On the existence of n-connected graphs with prescribed degrees ($n \geq 2$). *Networks*, 3:225–239, 1973.

[21] R.S. Wilkov. Analysis and design of reliable computer networks. In *Large-Scale Networks: Theory and Design*, F.T. Boesch (Editor), pp. 158–176, IEEE Press, 1976.