Alley CATs in search of good homes

Frank Ruskey [*]
Department of Computer Science
University of Victoria
Victoria, B.C., V8W 3P6, Canada

Robert Cohen and Peter Eades and Aaron Scott
Department of Computer Science
The University of Newcastle
Newcastle, NSW, Australia

Abstract

A score sequence of a tournament is a monotonically non-decreasing sequence of the out-degrees of its vertices. We develop two algorithms that generates all score sequences, each of which appears to use only a constant amount of computation per sequence produced.

A degree sequence of a graph is a monotonically non-increasing sequence of the degrees of its vertices. We develop an algorithm that generates all degree sequences of given length and observe that the algorithm does only constant computation per sequence produced. This algorithm is extended to generate degree sequences of connected graphs, and of biconnected graphs.

1 Introduction

There is considerable interest in the Mathematical community in the sequence of degrees of vertices of directed and undirected graphs. Indeed, a search of the MathSci Index reveals well over 200 references which contain "degree sequence" in the title or abstract. We consider here the question of efficiently generating these sequences.

When generating combinatorial sequences a desirable goal is that the running time of the algorithm be proportional to the number of sequences produced. Such algorithms are said to be CAT, meaning that they run in Constant Amortized Time. For the purposes of this paper we regard an algorithm as being an *alley CAT* if its experimental behavior is observed to be CAT, but that no proof of CAT behavior has been found. Once a proof has been given the algorithm has found a *good home* and is no longer an alley CAT. In this paper we present simple alley CATs for generating degree sequences of graphs and score sequence of tournaments.

[*]Research supported by NSERC grant A3379.

2 Score Sequences

A *tournament* is a directed graph $G = (V, E)$ in which, for every pair $u, v \in V$, either $(u, v) \in E$ or $(v, u) \in E$, but not both. In other words, it is a complete graph in which every edge has been given an orientation. A *score sequence* of a tournament is a monotonically non-decreasing sequence of the out-degrees of its vertices. For example, any transitive tournament on n vertices has score sequence $(0, 1, \ldots, n-1)$.

Algorithms for generating score sequences are given in the papers of Narayana, Mathsen, and Sarangi [11] and Gervacio [6]. Neither of these algorithms runs in constant amortized time nor are these as conceptually simple as the algorithm we present below.

There is a simple characterization of score sequences due to Landau [9].

Theorem 1 (Landau) *A sequence $s_1 \le s_2 \le \cdots \le s_n$ of integers is a score sequence of a tournament if and only if*

$$\sum_{i=1}^{k} s_i \ \ge\ \binom{k}{2} \tag{1}$$

for $k = 1, 2, \ldots, n$ with equality holding for $k = n$.

The previous theorem suggests that it may be possible to characterize those sequences that are prefixes of score sequences of some given length. Define a *n-prescore sequence* to be any prefix of some score sequence of length n. For example, 122 is a 5-prescore sequence but is not a 4-prescore sequence. Note that we wrote 122 for $(1, 2, 2)$ above; we will drop the use of commas and parentheses in our examples when no confusion can arise. Our algorithm is based on the following theorem, which follows easily from the previous theorem.

Theorem 2 *If $s_1, s_2, \ldots, s_{k-1}$ is a n-prescore sequence then $s_1, s_2, \ldots, s_{k-1}, s_k$ is a n-prescore sequence if and only if*

$$\max(s_{k-1}, \binom{k}{2} - p_k) \le s_k \le \left\lfloor \frac{\binom{n}{2} - p_k}{n-k+1} \right\rfloor,$$

where $p_k = s_1 + s_2 + \cdots + s_{k-1}$, and $s_0 = 0$.

This theorem leads immediately to the simple algorithm shown in Figure 1.

The initial call is `gen(1,0)` and `C[k]` is the binomial coefficient $\binom{k}{2} = k(k-1)/2$. Parameter `sum` is the same as the value p_k. The call `PrintIt( sum )` sets `s[n]` to `C[n]-sum`, and does whatever processing of the score sequence is desired.

Since only a constant amount of computation is done for each recursive call and every recursive call produces at least one score sequence, the amount of computation used by the algorithm is proportional to the number of nodes in the underlying computation tree.

Unfortunately, it seems difficult to determine the size of this tree since even determining the asymptotic number of leaves requires considerable effort. Let $s(n)$ denote

```
procedure gen ( k, sum : integer );
var i : integer;
begin
  if k = n then PrintIt( sum )
  else begin
     for i := max(s[k-1],C[k]-sum) to (C[n]-sum) div (n-k+1) do begin
        s[k] := i;  gen( k+1, sum+i );
     end;
  end;
end {of gen};
```

Figure 1: Pascal code for generating score sequences.

the number of score sequences of length n. Winston and Kleitman [18] show that there are constants c_1 and c_2 such that

$$c_1 \frac{4^n}{n^{5/2}} \; < \; s(n) \; < \; c_2 \frac{4^n}{n^2},$$

but the exact asymptotic value of $s(n)$ is unknown, although they conjecture that $s(n) = \Theta(4^n/n^{5/2})$. It seems plausible that their methods could be used to bound the size of the computation tree.

Nevertheless, Table 2 shows that the ratio of recursive calls to number of leaves is monotonically decreasing (for $n > 5$) to some limit less than three, and so the algorithm is an alley CAT.

2.1 Another approach

The approach of the previous section might be called the "increasing sequence approach". Another approach, the "integer partition approach", has been suggested by Carla Savage [14]. Define $\mathbf{P}_n(p,k)$ to be the set of all sequences $s_1 \le s_2 \le \cdots \le s_n$ that satisfy (1) and such that $\sum_{i=1}^n s_i = p$ and $s_n \le k$. The number of score sequences of length n is $P_n(\binom{n}{2}, n-1)$, where $P_n(p,k) = |\mathbf{P}(n,k)|$. We are led to the following recurrence relation, which has no zero terms (and therefore can be turned into a recursive algorithm where every call produces at least one score sequence).

$$P_n(p,k) \;=\; \begin{cases} 1 & \text{if } n = 1, k \ge p \\ P_{n-1}(p-k,k) & \text{if } k-1 < p/n \\ P_n(p,k-1) & \text{if } p-k < \binom{n-1}{2} \\ P_{n-1}(p-k,k) + P_n(p,k-1) & \text{otherwise.} \end{cases}$$

The progam that arises from this recurrence relation is shown in Figure 2. It is also an alley CAT, but the size of its computation tree is about twice as large as the computation tree of the original algorithm of Figure 1.

n	*1-nodes*	*calls*	$s(n)$	*calls* / $s(n)$
1	1	1	1	1.000000
2	2	2	1	2.000000
3	4	5	2	2.500000
4	7	10	4	2.500000
5	18	25	9	2.777777
6	42	60	22	2.727272
7	113	161	59	2.728813
8	311	445	167	2.664670
9	895	1283	490	2.618367
10	2657	3817	1486	2.568640
11	8139	11702	4639	2.522526
12	25522	36721	14805	2.480310
13	81599	117493	48107	2.442326
14	265493	382484	158808	2.408468
15	876985	1263896	531469	2.378118
16	2934862	4230834	1799659	2.350908
17	9934666	14324411	6157068	2.326498
18	33972892	48990040	21258104	2.304534
19	117229853	169058807	73996100	2.284698
20	407804236	588104519	259451116	2.266725

Table 1: The number of recursive calls from `gen(n)`.

```
procedure gen ( n,p,k : integer );
begin
  if n = 1 then begin  a[1] := p;  PrintIt  end
  else begin
     if k-1 >= ceiling( p,n ) then gen( n, p, k-1 );
     if p-k >= C[n-1] then begin
        a[n] := k;  gen( n-1, p-k, k );
     end;
  end;
end {of gen};
```

Figure 2: Another algorithm for generating score sequences.

3 Degree Sequences

Degree sequences have been characterized and several papers written about building graphs with given degree sequence, but, suprisingly, there seem to be no published algorithms for generating degree sequences. Our aim in this paper is to develop an efficient algorithm for generating all degree sequences. Our algorithm can be thought of as an instance of "reverse search", a term originated by Avis and Fukuda [2]. That is to say, we impose a rooted tree structure on the set of degree sequences by noting how to obtain the parent of a node, inverting the relationship to obtain the children of a node, and finally implementing an algorithm based on this inversion.

A sequence $D = (d_1, d_2, \ldots, d_n)$ is said to be *graphical* if there is a graph on vertex set $[n]$ such that vertex i has degree d_i. If in addition $d_1 \geq d_2 \geq \cdots \geq d_n$ then D is a *degree sequence*. Define $\mathbf{D}_n$ to be the set of all degree sequences of length n.

There are two simple characterization of degree sequences. The oldest, due to Havel [8] and Hakimi [7] turns out to be most useful for our purposes. Another characterization, due to Erdös and Gallai [4], one the surface appears similar to the Landau characterization of score sequences, but unfortunately does not seem to lend itself to a characterization of prefixes of degree sequences.

Theorem 3 (Erdös-Gallai) *Let $D = (d_1, d_2, \ldots, d_n)$ be a sequence of integers with $n > d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. Then D is a degree sequence if and only if, for $i = 1, 2, \ldots, n$,*

$$\sum_{j=1}^{i} d_i \;\leq\; i(i-1) + \sum_{j=i+1}^{n} \min\{i, d_j\}.$$

Theorem 4 (Havel-Hakimi) *A sequence $d_1 \geq d_2 \geq \cdots \geq d_n$ of integers is the degree sequence of some graph if and only if $c_1, c_2, \ldots, c_{n-1}$ is graphical, where*

$$c_i = \begin{cases} d_{i+1} - 1 & \text{if } 1 \leq i \leq d_1 \\ d_{i+1} & \text{if } d_1 + 1 \leq i \leq n-1 \end{cases}$$

Let $\mathbf{S}_n$ be the set of all sequences $D = (d_1, d_2, \ldots, d_n)$ for which $n - 1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$. Define a function $f_n : \mathbf{S}_n \to \mathbf{S}_{n-1}$ as follows. First, let $h = d_n$, $p = \max\{j \mid d_j > d_h\}$, and $q = \min\{j \mid j \leq n \text{ and } d_j < d_h\}$. Given $D \in \mathbf{S}_n$, define $C = (c_1, c_2, \ldots, c_{n-1}) = f_n(D)$ as follows.

$$c_i = \begin{cases} d_i - 1 & \text{if } 1 \leq i \leq p \text{ or } p + q - h \leq i \leq q - 1, \\ d_i & \text{if } p + 1 \leq i \leq p + q - h - 1 \text{ or } q \leq i \leq n - 1. \end{cases}$$

The subscript n in f_n will be dropped when no ambiguity can arise. We call $f(D)$ the *reduction* of D. Our algorithm is based on the following variant of Theorem 4 (from Takahashi, Imai and Asano [16]).

Corollary 1 *For $n > 0$ and $D \in \mathbf{S}_n$, the sequence $D \in \mathbf{D}_n$ if and only if $f_n(D) \in \mathbf{D}_{n-1}$.*

Note that f imposes an infinite tree structure on the set of all degree sequences. The root of the tree is the single degree sequence 0 of a graph with one vertex. Otherwise, degree sequence D has parent $f(D)$. Level n (with the root at level 1) in this tree consists of all degree sequences of length n.

In the theorem above let f denote the function that maps D to C; i.e., $f(D) = C$. This theorem implies that there is a tree structure of degree sequences where $f(D) = C$ is the parent of D. Of course, C will, in general, have children other than D. The root of the tree is the single element sequence (0). Theorem 1 tells us the parent of a node; to develop a generation algorithm we need to know all children of a node, that is, we wish to know the set $f^{-1}(C)$. A careful examination of Theorem 1 reveals the following characterization.

Theorem 5 *Let $C \in \mathbf{D}_{n-1}$. The sequence $D \in f^{-1}(C)$ if and only if one of the four mutually exclusive conditions given below are met. Let h denote d_n, $p = \max\{i \mid c_i > c_h \text{ or } i = 0\}$, and $q = \min\{j \mid c_j < c_h \text{ or } j = n\}$.*

[Case 1a: $h = c_{n-1} + 1$ and $c_h = h$ and $p + n - q \leq h$.]

$$d_i = \begin{cases} c_i + 1 & \text{if } 1 \leq i \leq h - n + q \text{ or } q \leq i \leq h \\ c_i & \text{if } h - n + q + 1 \leq i \leq q - 1 \end{cases}$$

[Case 1b: $h = c_{n-1} + 1$ and $c_h = h - 1$ and $s + n - p - 1 \leq h$, where $s = \max\{i \mid c_i > h\}$.]

$$d_i = \begin{cases} c_i + 1 & \text{if } 1 \leq i \leq h - n + p + 1 \text{ or } p + 1 \leq i \leq n - 1 \\ c_i & \text{if } h - n + p + 2 \leq i \leq p \end{cases}$$

[Case 2a: $0 \leq h \leq c_{n-1}$ and there are a, b such that $a \leq b$, $a + b = h + q$, $a > p$, $b < n$, and $c_b = c_h - 1$]

$$d_i = \begin{cases} c_i + 1 & \text{if } 1 \leq i \leq a - 1 \text{ or } q \leq i \leq b \\ c_i & \text{if } a \leq i \leq q - 1 \text{ or } b \leq i \leq n - 1 \end{cases}$$

[Case 2b: $0 \leq h \leq c_{n-1}$ and there are a, b such that $a < b$, $a + b = p + h + 1$, $j < q$, and $c_a = c_h + 1$]

$$d_i = \begin{cases} c_i + 1 & \text{if } 1 \leq i \leq a - 1 \text{ or } p + 1 \leq i \leq b \\ c_i & \text{if } a \leq i \leq p \text{ or } b + 1 \leq i \leq n - 1 \end{cases}$$

The following examples illustrate the four cases.

- [Case 1a:] $f(7744444) = 664433$ or $f(77766666666) = 6666666555$.
- [Case 1b:] $f(333333) = 33222$.
- [Case 2a:] $f^{-1}(87776665555) \supseteq \{988876655555, 988777655555, 987777755555\}$.
- [Case 2b:] $f^{-1}(86666665555) \supseteq$ $\{977766665555, 977666666555, 976666666655, 966666666665\}$.

Proof. Since $d_n \leq d_{n-1} \leq c_{n-1} + 1$, it is clear that $h = d_n$ must satisfy $0 \leq h \leq c_{n-1} + 1$. We now broadly classify the degree sequences according to whether $h = c_{n-1} + 1$ (Case 1) or $0 \leq h \leq c_{n-1}$ (Case 2). Throughout the proof we use y as a notation for c_h (especially in the diagrams).

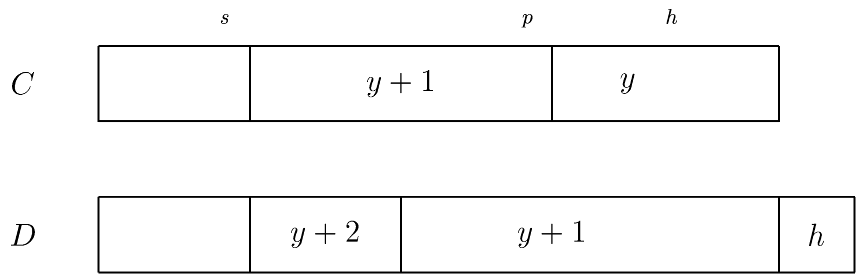
If $h = c_{n-1} + 1$, then $d_{n-1} = h$ (since c_{n-1} can be at most one less than d_{n-1}, and $d_{n-1} \neq h - 1$ because D is non-increasing). Since h becomes $h - 1$ in position n, it follows that either $c_h = h$ (Case 1a) or $c_h = h - 1$ (Case 1b).

[Case 1a:]

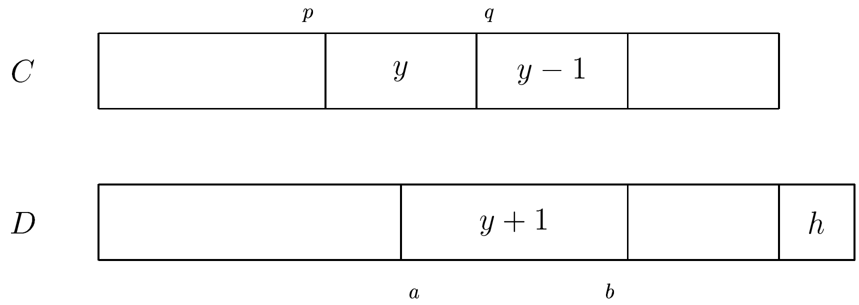
Here we must have $d_h = h$ and $c_q = h - 1$; there are thus $n - q$ elements equal to $h - 1$. Also, the elements in positions 1 thru p get reduced; the only other elements that can be reduced would appear in the range $p+1$ to $h-1$. To get h elements of D reduced in obtaining C we must therefore have $h \geq p + n - q$. On the other hand, if the last inequality holds, then the update given in the statement of the lemma does specify a D such that $f(D) = C$.

The proofs of the other cases follow by similar exhaustive reasoning and are omitted. We do provide some helpful diagrams.

[Case 1b:]



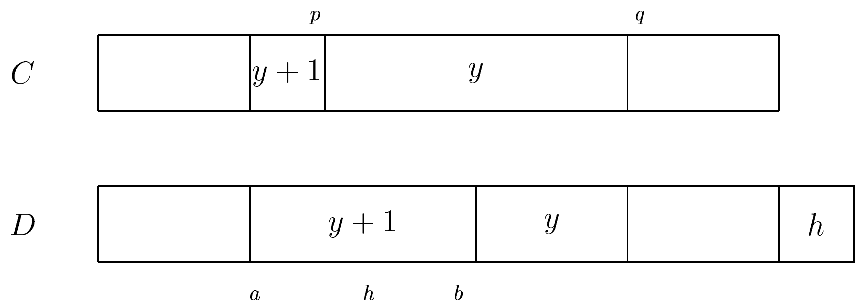
[Case 3:]



[Case 4:]

$n\backslash d_1$	0	1	2	3	4	5	6	7	8	9	10	11
1	1											
2	1	1										
3	1	1	2									
4	1	2	4	4								
5	1	2	7	10	11							
6	1	3	10	22	35	31						
7	1	3	14	34	78	110	102					
8	1	4	18	54	138	267	389	342				
9	1	4	23	74	223	503	968	1352	1213			
10	1	5	28	104	333	866	1927	3496	4895	4361		
11	1	5	34	134	479	1356	3417	7221	12892	17793	16016	

Table 2: Number of degree sequences classified by d_1.



□

Note that $|f^{-1}(C)| > 1$ unless C consists of all 0's.

We may implement Theorem 5 in a fairly straightforward manner using recursion. A Pascal implementation may be found in the Appendix. We have used the algorithm to compute a table of the number of degree sequences of length n with largest degree d (see Table 2). Table 3 shows the result of running the algorithm for $1 \leq n \leq 16$. The cost is the sum of the number of recursive calls together with a count of the number of times through all of the loops. This gives and accurate (up to constant factors as a function of n) measure of the amount of computation done by the algorithm. Since the cost is decreasing to some constant less than 4, we conclude that the algorithm is an alley CAT.

The enumeration of degree sequences appears to be at least as difficult as the enumeration of score sequences. See Stanley [15] and Peled [12] for some recent results on this question. It may therefore be rather difficult to obtain accurate statements about the cost as a function of n.

n	$\lvert\mathbf{D}_n\rvert$	$calls/\lvert\mathbf{D}_n\rvert$	$cost/\lvert\mathbf{D}_n\rvert$	$cost$
1	1	1.00000	0.00000	0
2	2	1.50000	2.50000	5
3	4	1.75000	5.00000	20
4	11	1.63636	5.00000	55
5	31	1.58065	5.35484	166
6	102	1.48039	4.65686	475
7	342	1.44152	4.53509	1551
8	1213	1.40643	4.21682	5115
9	4361	1.39119	4.13667	18040
10	16016	1.37881	4.02323	64436
11	59348	1.37209	3.97944	236172
12	222117	1.36661	3.92911	872722
13	836315	1.36296	3.89905	3260831
14	3166852	1.35994	3.86925	12253345
15	12042620	1.35762	3.84654	46322365
16	45967479	1.35567	3.82570	175857659

Table 3: The cost of generating degree sequences.

3.1 Degree Sequences of k-connected Graphs

There has been some interest in degree sequences of k-connected graphs. We say that a sequence is k-connected graphical if it there is some k-connected graph with that sequence as its sequence of degrees. The following theorem is to be found in Berge [3].

Theorem 6 *A degree sequence is a 1-connected degree sequence if and only if* $d_n \geq 1$ *and* $\sum_{i=1}^{n} d_i \geq 2(n-1)$.

More useful for our purposes is the following theorem of Asano [1].

Theorem 7 (Asano) *A degree sequence* D *is 1-connected graphical if and only if* $f(D)$ *is 1-connected graphical, or* $D = (1,1)$.

To apply Theorem 7 to the problem of generating all graphical sequences we need only change the lower limit of the outer loop from 0 to 1.

What about k-connectivity? Consider $D = 22222$ with $k = 2$. Since $f(D) = 2211$ is not 2-connected graphical, Theorem 7 cannot be extended to k-connectedness. The following theorem of Wang and Kleitman [17] generalizes Berge's result.

Theorem 8 (Wang-Kleitman) *A degree sequence* D *is* k*-connected graphical if and only if* $n \geq k+1$, $d_n \geq k$, *and*

$$\frac{1}{2}\sum_{j=1}^{n} d_j \;\geq\; n - \binom{k}{2} - 1 + \sum_{j=1}^{k-1} d_j$$

We observe the following corollary.

Corollary 2 *If D is a degree sequence with $d_n \geq k$, $n \geq k+1$, and $d_{n-2} \geq k+1$ then D is k-connected graphical.*

Proof. We show that

$$d_k + \cdots + d_n \geq 2n - k(k-1) - 2 + d_1 + \cdots + d_{k-1}$$

and then appeal to Theorem 8. Let p be such that $d_p > k$ and $d_{p+1} = k$. The condition of the corollary implies that $p \geq n-2$.

$$\begin{aligned} d_k + \cdots + d_n &\geq (k+1)(p-k+1) - k(n-p) \\ &\geq 2n - k(k-1) - 2 + (k-1)(n-1) \\ &\geq 2n - k(k-1) - 2 + d_1 + \cdots + d_{k-1} \end{aligned}$$

□

We now consider 2-connected graphical sequences.

Lemma 1 *Sequence D with $n \geq 3$ is 2-connected if and only if either $f(D)$ is 2-connected or $D = (2, 2, \ldots, 2)$.*

Proof. Assume that D is 2-connected but $C = f(D)$ is not. Either $c_{n-1} = 1$ or $c_{n-1} = 2$ by Corollary 2.

If c_{n-1}, then $d_{n-1} = d_n = 2$. If $d_1 = 2$, then $D = (2, 2, \ldots, 2)$ which is accounted for in the statement of the lemma. If $d_1 > 2$ then $D = (d, 2, 2, \ldots, 2)$ where $d \geq 4$; but this is not 2-connected by Theorem 8.

If $c_{n-1} = 2$, then by Corollary 2 $c_{n-3} = c_{n-2} = c_{n-1} = 2$. Now consider the two cases $d_n = 3$ and $d_n = 2$. If $d_n = 3$ then by the nature of $f(D)$ we have $d_1 = d_2 = \cdots = d_n = 3$, and $C = (3, 3, \ldots, 3, 2, 2, 2)$, which is 2-connected by Theorem 8. For $d_n = 2$ we now consider the two cases $c_1 = d_1 - 1$ and $c_1 = d_1$.

If $d_n = 2$ and $c_1 = d_1 - 1$, then

$$\begin{aligned} \sum_{i=1}^{n-1} c_i &= -2d_n + \sum_{i=1}^{n} d_i \qquad \text{(always holds)} \\ &= 2n - 8 + 2d_1 \\ &\geq 2n - 8 + 2(c_1 + 1) \\ &\geq 2(n-1) - 4 + 2c_1 \end{aligned}$$

If $d_n = 2$ and $c_1 = d_1$, then $d_1 = d_2 = d_3$. Now observe, since $d_3 > 2$ and $\sum_{i=1}^{n} d_i$ is even, that

$$d_3 + \cdots + d_n \geq 2(n-2) + 2,$$

and hence that

$$\begin{aligned} \sum_{i=1}^{n-1} c_i &= -4 + \sum_{i=1}^{n} d_i \\ &\geq -4 + 2n - 4 + 2 + 2d_1 \\ &= 2n - 6 + 2d_1 \\ &= 2(n-1) - 4 + 2c_1 \end{aligned}$$

□

4 Final Remarks

There are a number of other problems about generating various types of degree sequences that suggest themselves. We have solved some of these problems and are working on others. These results will be presented later in a more complete version of this paper.

One problem is to extend the results on connected and biconnected graphs to larger values of k-connectedness.

Another problem related to the ones considered in this paper is that of generating the corresponding sequences for digraphs that we called degree sequences for graphs. We define a sequence

$$(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p) \quad \text{with} \quad s_1 \geq s_2 \geq \cdots \geq s_p \tag{2}$$

to be *digraphical* if the vertices of some digraph can be labelled $1, 2, \ldots, n$ such that s_i is the outdegree of the vertex i and t_i is the indegree of vertex i. These sequences are characterized by the following theorem of Fulkerson [5] and Ryser [13].

Theorem 9 (Fulkerson-Ryser) *A sequence as in (2) is digraphical if and only if the following three conditions hold.*

(a) $s_i \leq p-1$ *and* $t_i \leq p-1$ *for* $i = 1, 2, \ldots, p$, *and*

(b) $\sum_{i=1}^{p} s_i = \sum_{i=1}^{p} t_i$, *and*

(c)

$$\sum_{i=1}^{n} s_i \leq \sum_{i=1}^{n} \min\{p-1, t_i\} + \sum_{i=n+1}^{p} \min\{n, t_i\} \quad \textit{for} \quad 1 \leq n < p.$$

Similarly, one may define bipartite graphical sequence pairs $\{S, T\}$, where S is the degree sequence of vertices in one partite set and T is the degree sequence of vertices in the other set. The following characterization is known (see Takahashi, Imai, and Asano [16]).

Theorem 10 *Let* $S = (s_1, s_2, \ldots, s_n)$ *and* $T = (t_1, t_2, \ldots, t_m)$ *be a pair of non-negative integer sequences with* $\sum_{i=1}^{n} s_i = \sum_{j=1}^{m} t_j$, $n \geq s_1 \geq s_2 \cdots \geq s_n$, $t_1 \leq t_2 \leq \cdots \leq t_m$. *Then* $\{S, T\}$ *is bipartite graphical if and only if, for all* $i = 1, 2, \ldots, n$ *and* $j = 1, 2, \ldots, m$,

$$\sum_{k=1}^{i} s_k \leq i(m-j) + \sum_{k=1}^{j} t_k$$

Instead of generating all degree sequences with n vertices, we could consider the problem of generating all degree sequences of graphs with m edges and no isolated vertices (to ensure finiteness). Let $g(m)$ be the number of such degree sequences and let $p(m)$ be the number of integer partitions of m. A long-standing conjecture is that the limit, as m tends to infinity, of $g(m)/p(m)$ is zero.

We have developed a representation of degree sequences that allows us to generate them in a CAT manner using essentially the same algorithm as that developed in this paper. This representation is a modified run-length encoding of the degree sequence.

5 Acknowledgement

The first author would like to thank the hospitality of the Department of Computer Science, University of Newcastle, Australia, where he spent his sabbatical and this research was conducted.

References

[1] T. Asano, "Graphical degree sequence problems with connectivity requirements", Lecture Notes in Computer Science #762, (1993) 38-47.

[2] D. Avis and K. Fukuda, "Reverse Search for Enumeration", *Discrete Applied Mathematics*, to appear.

[3] C. Berge, *Graphes et Hypergraphes*, Dunod, 1970.

[4] P. Erdös and T. Gallai, "Graphs with prescribed degrees of vertices (Hungarian)", *Mat. Lapok*, 11 (1960) 264-274.

[5] D.R. Fulkerson, " Zero-one matrices with zero traces", *Pacific J. Mathematics*, 10 (1960) 831-836.

[6] S.V. Gervacio, "Score Sequences: Lexicographic Enumeration and Tournament Construction", *Discrete Mathematics*, 72 (1988) 151-155.

[7] S. Hakimi, "On the realizability of a set of integers as degrees of the vertices of a graph", *SIAM J. Applied Math.*, 10 (1962) 496-506.

[8] V. Havel, "A remark on the existence of finite graphs (Hungarian)", *Časopis Pěst. Mat.*, 80 (1955) 477-480.

[9] H.G. Landau, "On dominance relations and the structure of animal societies, III. The condition for a score structure", *Bulletin for Mathematical Biophysics*, 15 (1953) 143-148.

[10] John W. Moon, *Topics on Tournaments*, Holt, Rinehart, and Winston, 1968.

[11] T.V. Narayana, R.M. Mathsen, and J. Sarangi, "An algorithm for Generating Partitions and its Applications", *J. Combinatorial Theory (A)*, 11 (1971) 54-61.

[12] U.N. Peled and M.K. Srinivasan, *The polytope of degree sequences*, Linear Algebra Appl., 114 (1989) 349-377.

[13] H. Ryser, *Combinatorial Mathematics*, Carus Mathematical Monographs, MAA, 1963.

[14] C. Savage, *personal communication.*

[15] R.P. Stanley, *A Zonotope Associated with Graphical Degree Sequences*, manuscript, 1989, appeared in *Applied Geometry and Discrete Math.*, 1991, Dimacs Series, AMS.

[16] M. Takahashi, K. Imai, and T. Asano, "Graphical Degree Sequence Problems", Tech. Report TR93-AL-33-15, Information Processing Society of Japan, (1993) 111-118.

[17] D.L. Wang and D.J. Kleitman, "On the existence of n-connected graphs with prescribed degrees", *Networks*, 3 (1973) 225-239.

[18] K.J. Winston and D.J. Kleitman, "On the asymptotic number of tournament score sequences", J. Combinatorial Theory (A), 35 (1983) 208-230.

6 Appendix

In this Appendix we present our Pascal code to generate degree sequences.

```
program DegreeSequence;

var
  d : array [0..100] of integer;
  n,k,i : integer;

procedure PrintIt( n : integer );
var i : integer;
begin
  for i := 1 to n do write( d[i]:2 );
  writeln;
end {of PrintIt};

procedure gen ( k : integer );
var h,p,q,s,i,j,hval : integer;
begin
  if k > n then PrintIt( n )
  else begin
     h := d[k-1] + 1;
     hval := d[h];
     p := h;  while d[p] = hval do p := p - 1;
     q := h;  while d[q] = hval do q := q + 1;
     if q > k then q := k;
     {------ Case 1a: ------}
     if d[h] = h then begin
        if p+k-q <= h then begin
           for i := 1 to h-k+q {p} do d[i] := d[i] + 1;
           for j := q to k-1 do d[j] := d[j] + 1;
           d[k] := h;  gen( k+1 );
           for i := 1 to h-k+q {p} do d[i] := d[i] - 1;
           for j := q to k-1 do d[j] := d[j] - 1;
        end;
     end;
     {------ Case 1b: ------}
     if d[h] = h-1 then begin
        s := p;
        while d[s] = h do s := s-1;
        if s < 0 then s := 0;
        if s+k-p-1 <= h then begin
```

```
          for i := 1 to h-k+p+1 do d[i] := d[i] + 1;
          for j := p+1 to k-1 do d[j] := h;
          d[k] := h;  gen( k+1 );
          for i := 1 to h-k+p+1 do d[i] := d[i] - 1;
          for j := p+1 to k-1 do d[j] := h - 1;
        end;
      end;
      {------ Case 2: ------}
      for h := d[k-1] downto 0 do begin
        hval := d[h];
        p := h;  while d[p] = hval do p := p - 1;
        q := h;  while d[q] = hval do q := q + 1;
        if q > k then q := k;
        d[k] := h;
        for i := 1 to h do d[i] := d[i] + 1;
        gen( k+1 );
        i := h;  j := q;
        while (i > p) and (d[j] = hval - 1) and (j < k) do begin
          d[j] := d[j] + 1;  j := j + 1;
          d[i] := d[i] - 1;  i := i - 1;
          gen( k+1 );
        end;
        for j := j-1 downto q do d[j] := d[j] - 1;
        for i := i+1 to h do d[i] := d[i] + 1;
        i := p;  j := h+1;
        while (j < q) and (d[i] = d[i+1]+1) do begin
          d[i] := d[i] - 1;  i := i - 1;
          d[j] := d[j] + 1;  j := j + 1;
          gen( k+1 );
        end;
        for i := i+1 to p do d[i] := d[i] + 1;
        for j := j-1 downto h+1 do d[j] := d[j] - 1;
        for i := 1 to h do d[i] := d[i] - 1;
      end;
  end;
end {of gen};

begin
  write( 'Enter n,k: ' ); readln( n,k );
  if k > 1 then begin
    write( 'Enter d[1..k-1]: ' );
    for i := 1 to k-1 do read( d[i] );
    readln;
  end;
  d[0] := n+1;
  gen( k );
end.
```