Generating Graph From Given Degree Sequences

Abhishek, Laura, Ravi

January 24, 2007

Abstract

We wish to find if we can generate a graph having red and blue colored edges, whose vertices satisfy 2 degree sequences, each corresponding to the red edges and blue edges respectively.

1 Introduction

Definition 1: A finite sequence $(d_1, d_2, \ldots, d_n)$ of nonnegative integers is called graphic (or realizable) if there is a labeled simple graph with vertex set $(v_1, v_2, \ldots, v_n)$ in which vertex v_i has degree d_i. Such graph is called realization of the given degree sequence $(d_1, d_2, \ldots, d_n)$.

We intend to test whether a given degree sequence is graphic. A simple recursive algorithm to test if the degree sequence is graphic was developed independently by Havel and Hakimi. We state their results in following theorem.

Theorem 1: Let **d**$= (d_1, d_2, \ldots, d_n)$ be a non increasing sequence of nonnegative integers $(n \geq 2)$ and denote the sequence $(d_2-1, d_3-1, \ldots d_{d_1+1}-1, d_{d_1+2}-1, \ldots, d_n) =$ **d'**. Then **d** is graphic if and only if **d'** is graphic.

Proof: It is immediate that if **d'** is graphic then **d** is graphic. Take a realization of **d'** with vertices $v_2, v_3, \ldots, v_n$. Introduce a new vertex v_1 and join v_1 to the d_1 vertices whose degree had 1 subtracted from them.

Now, assume **d** is graphic and let G with vertices $(v_1, v_2, \ldots, v_n)$ be realization of **d**. If vertex v_1 is connected to vertices $v_2, v_3, \ldots, v_{d_1+2}$, we are done, since we can delete v_1 and the corresponding d_1 edges adjacent to v_1

from G. Assume that there exists a vertex v_x which is adjacent to v_1 and v_x doesnt belong to $v_2, v_3, \dots, v_{d_1+2}$. Let $v_y \in v_2, v_3, \dots, v_{d_1+2}$ such that v_1 and v_y are not adjacent. If $degree(v_x) = degree(v_y$, we can interchange vertices v_x and v_y wihout affecting the degrees. If $degree(v_x) < degree(v_y)$, then there is a vertex $v_z \neq v_x$ joined by an edge to v_y but not to v_x. Perform a *switch*, by adding the edges (v_1, v_y), (v_z, v_x) and deleting the edges (v_1, v_x), (v_y, v_z). This doesn't affect the degrees. So, we still have the realization of **d**. Repeating this we obtain a realization of **d** where vertex v_1 is joined to d_1 highest degree vertices other than v_1 itself.

Theorem 1 thus gives a recursive test for whether d is graphic: apply the theorem repeatedly until either the theorem reports that the sequence is not graphical (if there are not enough vertices available to connect to some vertex) or sequence becomes zero vector (in which case **d** is graphic).

A Polynomial Time Algorithm via Matchings

For $\mathbf{d} = (d_1, d_2, \dots, d_n)$ as above, let M_d be the following graph. For each $1 \leq i \leq n$, M_d contains a complete bipartite graph $H_i = (L_i, R_i)$, where $|R_i|=n-1$ and $|L_i| = n - 1 - d_i$. The vertices of R_i are labeled so that there is a label for each $1 \leq j \leq n$ other than i; let us denote these labels by $u_{i,1}, \dots, u_{i,i-1}, u_{i,i+1}, \dots, u_{i,n}$. In addition, for each $1 \leq i, j \leq n$ with $j \neq i$, M_d has an edge between $u_{i,j}$ and $u_{j,i}$. Now, each perfect matching M of M_d gives rise to a unique realization G of **d** in the natural way: G has a link between v_i and v_j if and only if M contains the edge between $u_{i,j}$ and $u_{j,i}$.

2 Discussed 1/22/07

Notation:

- $d(v)$ = the degree of node v.

- $B(v)$ = the set of edges adjacent to node v.

- The *combinatorial algorithm* is the algorithm described above. It sorts the nodes in non-increasing order by degree, then connects the first node, v, in the sorted list to the next $d(v)$ nodes in the sorted list (ties are broken arbitrarily). The residual degrees are updated, the nodes are re-sorted, and the algorithm repeats until all nodes have the required number of edges or else there are not enough nodes left to fill the residual degree of some node.

- In the combinatorial algorithm, a node is *processed* when it is the highest degree node remaining and is connected to the appropriate ndoes.
- In the combinatorial algorithm, let $t(v) =$ the time at which node v is processed.
- In the combinatorial algorithm, let $d_{t(v)}(u) =$ the residual degree of node u right before node v is processed (i.e., at time $t(v)$).

Theorem 2.1. *Show that a solution to the following LP implies existance of an integral solution.*

1. $$\sum_{v \in V} d(v) \textit{ is even}$$

2. $$\forall v \in V, \sum_{e \in B(v)} x_e = d(v)$$

3. $$\forall e \in E, 0 \leq x_e \leq 1$$

Lemma 2.1. *If x is the first node that could not be given enough edges by the combinatorial algorithm, then $\forall$ edges (u, v) created by the combinatorial algorithm before x is processed, (u, x) has been created and/or (v, x) has been created.*

Proof. Assume the lemma is false. Let (u, v) be the edge created by the combinatorial algorithm such that (u, x) does not exist, (v, x) does not exist, and v is the last node processed by the algorithm that does not have an edge to x.

Now consider what happens when node v is processed by the combinatorial algorithm.

Suppose v is connected to k nodes: $p_1, p_2, \ldots, p_k$ that are not processed by the algorithm before x. Then, at this iteration, v must be connected to $d_{t(v)}(v) - k$ nodes $q_1, q_2, \ldots, q_{d_{t(v)}(v)-k}$ that are handled before x.

In other words, $t(q_i) < t(x) \forall i \in \{1, \ldots, k\}$, and $t(p_i) > t(x) \forall i \in \{1, \ldots, d_{t(v)}(v) - k\}$.

By definition of v, all w with $t(v) < t(w) < t(x)$ will be connected to x. This gives:

$$d_{t(v)}(x) - d_{t(x)}(x) \geq d_{t(v)}(v) - k \quad (1)$$
$$\Rightarrow \quad d_{t(v)}(v) \leq d_{t(v)}(x) - d_{t(x)}(x) + k \quad (2)$$

Is it possible that one of the k nodes p_i (connected to v) has $d_{t(x)}(p_i) = 0$? (that is, can any of these k nodes be "filled" before x is processed?)

If so, we have $d_{t(x)}(p_i) = 0$. Now, $d_{t(v)}(p) \geq d_{t(v)}(x) = d_{t(v)+1}(x)$, and $d_{t(v)+1}(p_i) = d_{t(v)}(p_i) - 1$. Therefore, $d_{t(v)+1}(p_i) + 1 \geq d_{t(v)+1}(x)$. Also, for all w such that $t(v) < t(w) \leq t(x)$,

$$\begin{aligned}
& d_{t(w)+1}(x) = d_{t(w)}(x) - 1 \\
\Rightarrow \quad & d_{t(w)}(p_i) + 1 \geq d_{t(w)}(x) \\
\Rightarrow \quad & d_{t(x)}(p_i) + 1 \geq d_{t(x)}(x) \geq 1 \\
\Rightarrow \quad & d_{t(x)}(x) = 1
\end{aligned}$$

So, right before x is processed, x has degree 1, but when x is handled, there are no nodes left with any residual degree. So the total residual degree of the algorithm is odd, so this case does not apply to this lemma. So we can say that $d_{t(x)}(p_i) > 0, \forall p_i$, and when x is processed, edges (x, p_i) can be created for $p_1, p_2, \ldots, p_k$. Since x is by definition not processed completed, we can say that

$$d_{t(x)}(x) > k \quad (3)$$

By **??** and **??**,

$$d_{t(v)}(v) \leq d_{t(v)}(x) - d_{t(x)}(x) + k < d_{t(v)}(x) - k + k = d_{t(v)}(x)$$

This contradicts the fact that $t(v) < t(x)$. So for all edges (u, v) created by the combinatorial solution before x is processed, edge (u, x) is also created and/or edge (v, x) is created. □

Lemma 2.2. *Suppose a fractional solution exists to the LP described above for some degree sequence, and the combinatorial algorithm cannot find an integral solution for the same degree sequence. Then, if x is the first node which the combinatorial algorithm cannot process completely, there exists some edge (u, v) created by the combinatorial algorithm (created before x is processed) such that at least one of u and/or v is not connected to x.*

Proof. ??? Scanty intuition: If such an edge exists, then we know we can convert this invalid integral solution into a fractional solution (by "stealing" some of that edge and giving it to node x). However, we need the "only if" side of that statement, which may or may not be true. □

Proof. (Theorem **??**) Follows directly from Lemma **??** and Lemma **??** □