Connected Realizations of Joint-Degree Matrices

Georgios Amanatidis* Bradley Green† Milena Mihail ‡

Abstract

We study a restriction of the classic degree sequence graphic realization problem studied by Erdős, Gallai, Havel, and Hakimi, namely the *joint-degree matrix graphic realization problem*. Here, in addition to the degree sequence, a joint degree matrix is given, the (i, j)th element of which specifies the exact number of edges between vertices of degree d_i and vertices of degree d_j. The decision and construction versions of the problem have a relatively straightforward solution. In this work, however, we focus on the corresponding *connected* graphic realization version of the problem. We give a necessary and sufficient condition for a connected graphic realization to exist, as well as a polynomial time construction algorithm that involves a novel recursive search of suitable local graph modifications. As a byproduct, we also suggest an alternative polynomial time algorithm for the joint-degree matrix graphic realization problem that never increases the number of connected components of the graph constructed.

Keywords: Joint-Degree Matrix, Graphic Realizations.

1 Introduction

Let $d_1 \geq d_2 \geq \ldots \geq d_n$ be a sequence of integers. The classic *graphic realization* problem asks whether there exists a simple graph on n vertices whose degrees are exactly $d_1, \ldots, d_n$. Erdős and Gallai showed that the natural necessary conditions for graphic realizability, namely that each subset of the highest k degree vertices can absorb their degrees within their subset and the degrees of the remaining vertices: $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min\{k, d_i\}$, are also sufficient [1, 2]. The well known Havel-Hakimi algorithm [3, 4] achieves such a realization in an efficient greedy way. It repeatedly sorts the vertices according to residual unsatisfied degree, picks any vertex of residual degree d_i, and connects it to the d_i vertices of highest residual degree. The process is repeated until all the degrees are satisfied. Further, if one wants to construct a *connected* graphic realization—an important requirement in networking—Erdős and Gallai showed that the obvious necessary condition $\sum_{i=1}^{n} d_i \geq 2(n-1)$ (i.e., there is a spanning tree) is also sufficient. In particular, it is easy to see that a non connected realization can be transformed to a connected one by a sequence of edge flips, each flip breaking a cycle inside a connected component and reducing the number of connected components by one. A

*School of Mathematics, Georgia Institute of Technology, USA.
Present address: Centrum Wiskunde & Informatica, The Netherlands. Email: georgios.amanatidis@cwi.nl
†School of Mathematics, Georgia Institute of Technology, USA.
Present address: Facebook, USA. Email: Brg@fb.com
‡College of Computing, Georgia Institute of Technology, USA. Email: mihail@cc.gatech.edu

flip picks two edges xy and uv such that xu and yv are non edges, removes xy and uv from the graph, and adds xu and yv. Clearly, flips do not change the degrees of the graph.

Here we study a restriction of this problem motivated by the need to generate graphs with realistic topologies, used to simulate network protocols and predict network evolution. Mahadevan et al. [5, 6], argued that a determining metric for a graph of given degrees to resemble a real network topology, is the specific number of links between vertices in different degree classes. Using heuristics that presumably approximate the target number of edges between degree classes, [6] constructed graphs strikingly similar to real network topologies. More recently, Gjoka et al. [7] proposed heuristics that also achieve other desired attributes, not captured from the number of links between vertices of different degrees, like the average clustering coefficient. Amanatidis et al. [8] formalized the approach of [5, 6] and defined the problem as we study it here. In that work, necessary and sufficient conditions for the existence of such graphs were given, as well as polynomial time constructions, some first results on sampling, and also a preliminary version of the material presented here.

Let $V = [n]$ be a set of vertices, $\mathcal{P}(V) = \{V_1, V_2, \ldots, V_k\}$ be a partition of V denoting subsets of vertices with the same degree, and $d : \mathcal{P}(V) \to \mathbb{N}$ be a function denoting the degree of vertices in V_i. For simplicity, we write $\mathcal{P}$ and d_i instead of $\mathcal{P}(V)$ and $d(V_i)$ respectively.[1] Also, let $D = (d_{ij})$ be a $k \times k$ matrix specifying the number of edges between V_i and V_j; if $i = j$, d_{ii}) is the number of edges entirely within V_i. We call such a D a *joint-degree matrix* (JDM for short). Note that d is implied by $\mathcal{P}$ and D through $d_i = \big(2d_{ii} + \sum_{j\in[k], j\neq i} d_{ij}\big)/|V_i|$, for $1 \leq i \leq k$.

In what follows, we assume that, given $\mathcal{P}$ and D, d is defined as above.

Definition 1. *JDM Graphic Realization*: Given $\mathcal{P}$ and D, decide whether there is a simple graph G on V, such that
(i) each vertex in V_i has degree d_i, $\forall i \in [k]$,
(ii) there are exactly d_{ij} edges between V_i and V_j, $\forall i, j \in [k]$ with $i \neq j$, and
(iii) there are exactly d_{ii} edges entirely inside V_i, $\forall i \in [k]$.
We use $\langle \mathcal{P}, D \rangle$ to denote the set of all such graphs.

Apart from [8], several other recent works [9, 10, 7] give the necessary and sufficient conditions below for $\langle \mathcal{P}, D \rangle$ to be nonempty, as well as polynomial time algorithms for constructing a graph in $\langle \mathcal{P}, D \rangle$. It should be noted here that Patrinos and Hakimi studied the same problem first [11], though with a somewhat different formulation.

Theorem 1 ([11, 8, 9, 10, 7])**.** *The following conditions are necessary and sufficient for the instance $\mathcal{P}$, D to have a graphic realization:*

1. Degree feasibility*:* $\big(2d_{ii} + \sum_{j\in[k], j\neq i} d_{ij}\big)/|V_i| = d_i \in \mathbb{N}$*, for* $1 \leq i \leq k$*.*
2. Matrix feasibility*: The matrix D is symmetric with nonnegative integral entries, and* $d_{ij} \leq |V_i| \cdot |V_j|$*, for* $1 \leq i < j \leq k$*, while* $d_{ii} \leq |V_i| \cdot (|V_i| - 1)/2$*, for* $1 \leq i \leq k$*.*

Theorem 1 is relatively straightforward, and the same holds for the proposed construction algorithms. Here, however, we focus on the corresponding *connected* graphic realization version of the problem:

[1] In the related literature it is often the case that $d(V_i) = i$. This is not necessary for any of our results. In particular, it is allowed to have $i \neq j$ such that $d(V_i) = d(V_j)$.

JDM Connected Graphic Realization: Given $\mathcal{P}$ and D, can we decide whether there exists a simple connected graph $G \in \langle \mathcal{P}, D \rangle$? Can we efficiently construct such a graph?

In Section 3 we give a necessary and sufficient condition for a connected graphic realization to exist (Theorem 5). Although the condition suggests checking a simple inequality in exponentially many graphs, we provide a polynomial time construction algorithm for the problem (Algorithm 4). Our algorithm, that involves a recursive search of suitable local graph modifications (Algorithm 3), returns either a graph in $\langle \mathcal{P}, D \rangle$, or a short certificate that no such graph exists. A necessary subroutine of our algorithm is an alternative polynomial time algorithm for JDM Graphic Realization, presented in Section 2, that never increases the number of connected components of the graph constructed.

2 An Alternative Algorithm for JDM Graphic Realization

Although the JDM Graphic Realization problem has a straightforward solution, this is not the case if we also ask for the resulting graph to be connected. In order to give the bigger picture, think of all of the constraints as upper bounds. That is, let $[\mathcal{P}, D] \supseteq \langle \mathcal{P}, D \rangle$ be the set of graphs such that for any $i, j \in [k]$ there are at most d_{ij} edges between V_i and V_j, there are at most d_{ii} edges inside V_i, and the degree of any vertex in V_i is upper bounded by d_i. The main idea is to construct a connected graph $G_0 \in [\mathcal{P}, D]$ and then grow it into a connected graph $G \in \langle \mathcal{P}, D \rangle$.

Towards this direction, we present an alternative algorithm for the JDM Graphic Realization problem. In particular, Algorithm 1, given any $G_0 \in [\mathcal{P}, D]$ as input, iteratively increases the number of edges by one at a time by appropriately adding and removing a few edges. Despite the fact that some existing edges may be removed in each iteration, the algorithm always makes sure that the number of connected components of the current graph is not increased. Moreover, after each iteration, the resulting graph is always in $[\mathcal{P}, D]$. In time polynomial in n, it outputs some $G \in \langle \mathcal{P}, D \rangle$, if such a graph exists.

When we just need to construct any graph in $\langle \mathcal{P}, D \rangle$, the natural choice of G_0 is the empty graph on V. However, in order to construct a connected $G \in \langle \mathcal{P}, D \rangle$, one should be more careful about the choice of the input. In particular, it would suffice to start with a tree in $[\mathcal{P}, D]$. The construction of such a tree is non trivial, and is the main focus of Section 3.

To facilitate the presentation, the procedure that increases the number of edges by one in each iteration of Algorithm 1 is stated separately. Recall that k is the number of vertex subsets in $\mathcal{P}$ and d is the (uniquely determined by $\mathcal{P}$ and D) function denoting the degree of vertices in each of this subsets.

Algorithm 1: JDM-Construct$(G_0, \mathcal{P}, d, D)$

```
1 G ← G_0
2 for i = 1 to k do
3     for j = i to k do
4         while the d_ij constraint is not saturated do
5             Augment(G, i, j, d, P)
6 return G
```

In the procedure *Augment* (Algorithm 2) the cases $i = j$ and $i \neq j$ are treated together since they differ only slightly. Of course, some conditions, like $u \neq v$ in line 1, are redundant when $i \neq j$. Here, and throughout this paper, we write xy for the edge $\{x, y\}$ for ease of notation.

Algorithm 2: Augment$(G, i, j, d, \mathcal{P})$

```
1  if ∃ u ∈ V_i, v ∈ V_j with u ≠ v, uv ∉ E(G), d(u) < d_i, d(v) < d_j then
2      Add the edge uv to E(G)
3  else if ∃ u ∈ V_i, v ∈ V_j with u ≠ v, uv ∉ E(G), d(u) < d_i, d(v) = d_j then
4      Pick a v' ∈ V_j with d(v') < d_j and, if possible, v' ≠ u
5      Find a neighbor x of v such that v'x ∉ E(G)
6      Remove the edge vx from E(G) and add the edges uv and v'x
7  else if ∃ u ∈ V_i, v ∈ V_j with u ≠ v, uv ∉ E(G), d(u) = d_i, d(v) < d_j then
8      Pick a u' ∈ V_i with d(u') < d_i and, if possible, u' ≠ v
9      Find a neighbor x of u such that u'x ∉ E(G)
10     Remove the edge ux from E(G) and add the edges uv and u'x
11 else
12     Pick u ∈ V_i, v ∈ V_j with u ≠ v, uv ∉ E(G)
13     Pick a v' ∈ V_j with d(v') < d_j
14     Find a neighbor x of v such that v'x ∉ E(G)
15     Remove the edge vx from E(G) and add the edge v'x
16     Pick a u' ∈ V_i with d(u') < d_i and, if possible, u' ≠ v
17     Find a neighbor y of u such that u'y ∉ E(G)
18     Remove the edge uy from E(G) and add the edges uv and u'y
```

The main underlying idea of Algorithm 2 is that as long as there exist unsaturated constraints, it is possible to get closer to a graph in $G \in \langle \mathcal{P}, D \rangle$ by adding (and possibly deleting) a very small number of edges. The same idea is used in other works that derive Theorem 1, e.g., [10, 7]. What makes Algorithm 2 somewhat more involved is the extra property of not affecting the connectivity.

Theorem 2. *If the degree and matrix feasibility conditions of Theorem 1 hold and $G_0 \in [\mathcal{P}, D]$, then Algorithm 1 constructs a graph $G \in \langle \mathcal{P}, D \rangle$ in time polynomial in n. Moreover, no iteration increases the number of connected components of the current graph.*

Proof. In what follows, we call an edge between V_i and V_j an ij-edge and, respectively, an edge inside V_i an ii-edge. Notice that every graph $G \in [\mathcal{P}, D]$ that satisfies all the edge constraints with equality, belongs to $\langle \mathcal{P}, D \rangle$ as well. To prove the theorem, it suffices to prove that given a graph $G \in [\mathcal{P}, D]$ with $\delta_{ij} < d_{ij}$ ij-edges for some $i, j \in [k]$, a single call of Augment$(G, i, j, d, \mathcal{P})$ returns a graph $G' \in [\mathcal{P}, D]$ with $\delta_{ij} + 1$ ij-edges, without affecting any other edge constraints and without increasing the number of connected components.

Together with the fact that $G_0 \in [\mathcal{P}, D]$, this implies that the while loop in line 4 of Algorithm 1 runs for at most d_{ij} iterations to saturate the corresponding constraint. Therefore, with no more than m calls of Augment$(\,\cdot\,)$—where m is the number of edges—Algorithm 1 ends up with a graph $G \in [\mathcal{P}, D]$ that satisfies all the edge constraints with equality, i.e.,

$G \in \langle \mathcal{P}, D \rangle$. It is straightforward to see that even a naive implementation of Algorithm 2 has a running time that is $O(n^2)$. Thus, the running time of Algorithm 2 is $O(m \cdot n^2)$.

So, consider the call of Augment$(G, i, j, d, \mathcal{P})$ on a graph $G \in [\mathcal{P}, D]$ with $\delta_{ij} < d_{ij}$ ij-edges, and let G' be the resulting graph. When the condition in line 1 holds, then clearly G' has $\delta_{ij} + 1$ ij-edges and no other edge constraints are affected, thus $G' \in [\mathcal{P}, D]$. Moreover, the number of connected components is not increased. So, suppose that the condition in line 1 does not hold.

If the condition in line 3 holds, then there exists some $v' \in V_j$ with $d(v') < d_j$, or else all the constraints associated with V_j would already be saturated. First, assume $v' \neq u$, and therefore $uv' \in E(G)$, or the condition in line 1 would hold. Since $d(v') < d(v)$ there exists some neighbor x of v such that $v'x \notin E(G)$. By removing the edge vx from $E(G)$ and by adding the edges uv and $v'x$, the degrees of only u and v' increase by one, the ij-edges increase by one, and no other edge constraints are affected; thus $G' \in [\mathcal{P}, D]$. To see that the number of connected components did not increase, recall that $uv' \in E(G)$, and notice that the only edge removed is vx. Nevertheless, v and x remain connected through the path (v, u, v', x).

Next, consider the case where $v' = u$, i.e., $i = j$ and there are no vertices in V_i with degree less than d_i other than u. Then $d(u) \leq d_i - 2$, since otherwise the corresponding degree feasibility condition of Theorem 1 would fail. Like before, $d(u) < d(v)$ and thus there exists some neighbor x of v such that $ux \notin E(G)$. By removing vx from $E(G)$ and by adding uv and ux, the degree of only u increases by two, the ii-edges increase by one, and no other edge constraints are affected; thus $G' \in [\mathcal{P}, D]$. Again the number of connected components is not increased, since the only edge removed is vx, but v and x remain connected through the path (v, u, x). So, suppose that the condition in line 3 does not hold.

If the condition in line 7 holds, the analysis is symmetric to the above. So, finally, suppose that the condition in line 7 does not hold either.

Due to the matrix feasibility conditions, there exist $u \in V_i, v \in V_j$ with $u \neq v$ and $uv \notin E(G)$, and it must be the case where $d(u) = d_i$, $d(v) = d_j$, or one of the conditions in lines 1, 3, and 7 would hold. Since not all the constraints associated with V_j are saturated, there exists some $v' \in V_j$ with $d(v') < d_j = d(v)$. Hence, there exists some neighbor x of v such that $v'x \notin E(G)$. By removing vx from $E(G)$ and by adding $v'x$—call G'' this intermediate graph—the degree of v' increases by one, the degree of v decreases by one, and no edge constraints are affected; thus $G'' \in [\mathcal{P}, D]$. Now, what happens in lines 16-18 is identical to what happens in lines 7-10, thus resulting in a graph $G' \in [\mathcal{P}, D]$ with $\delta_{ij} + 1$ ij-edges, without affecting any other edge constraints.

Here, to see that the number of connected components did not increase, first notice that $uv' \in E(G)$, otherwise the condition in line 7 would hold for G. Moreover, either $u' = v$ or $u'v \in E(G)$, otherwise the condition in line 3 would hold for G. So, in G' where vx and uy are removed, v and x remain connected through the path (v, u, v', x), while u and y remain connected either through the path (u, v, y) (when $u' = v$), or through the path (u, v, u', y) (when $u' \neq v$ and $u'v \in E(G)$). □

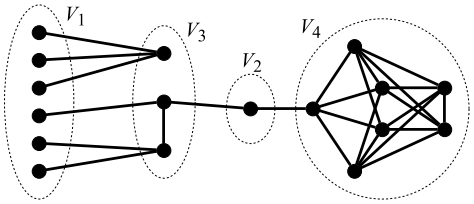
3 JDM Connected Graphic Realization

As already stated, the actual goal is to either construct a connected realization in $\langle \mathcal{P}, D \rangle$, or show that such a realization does not exist. Here it is not reasonable to expect that a simple condition like the Erdős-Gallai condition, i.e., the degrees summing up to at least $2(n-1)$, would suffice. In particular, there are graphically realizable instances with $\Theta(n^2)$ edges that have no connected realization. A straightforward such example is when all the edges are required to be inside distinct V_is, e.g., for $i = 1, 2$, V_i contains $n/2$ vertices of degree $n/2 - 1$ and D requires $\binom{n/2}{2}$ edges within each V_i but no edges between them.

These, however, are not the only problematic cases. Suppose that we have 6 vertices of degree 1 in V_1, 1 vertex of degree 2 in V_2, 3 vertices of degree 3 in V_3 and 7 vertices of degree 5 in V_4. The edge requirements are given by

$$D = \begin{pmatrix} 0 & 0 & 6 & 0 \\ 0 & 0 & 1 & 1 \\ 6 & 1 & 1 & 0 \\ 0 & 1 & 0 & 17 \end{pmatrix}.$$

There are a few such graphs and all look, more or less, like the graph below:



In such a case, although there seem to be enough edges in total, there are not enough edges outside of V_4 in order to connect everything else. The obvious observation—and the intuition behind our connectivity condition—is that we should think of V_4 as a single vertex in such a case; then it becomes obvious that there are too few edges. In fact, one can think of this as a certificate that this instance does not have a connected realization. Unfortunately, in more complex examples it is not at all clear which groups of vertices one should contract to get such a counterexample when one exists.

It is also easy to see that arbitrary simple flips cannot be used to decrease the number of connected components of a non-connected graphic realization $G \in \langle \mathcal{P}, D \rangle$, even when connected realizations do exist. In particular, let $uv, xy \in E(G)$, $ux, vy \notin E(G)$, such that $u \in V_i$, $v \in V_j$, $x \in V_{i'}$ and $y \in V_{j'}$. Note that the flip of removing uv and xy and adding ux and vy yields a graph in $\langle \mathcal{P}, D \rangle$ if and only if $V_i = V_{j'}$ or/and $V_j = V_{i'}$. Even then, however, the number of connected components may increase!

In this section we give a necessary and sufficient condition for the instance $\mathcal{P}$, D to have a connected realization. The proof also provides a polynomial time algorithm that constructs a connected realization, if one exists, or produces a certificate that $\mathcal{P}$, D does not have such a realization.

3.1 Reducing to a Weaker Problem on Another Instance

Eventually, we are going to show that the existence of connected graphs in $\langle \mathcal{P}, D \rangle$ is equivalent to the existence of a certain type of trees for a slightly simpler instance $\tilde{\mathcal{P}}$, $\tilde{D}$. In this

subsection we define $\tilde{\mathcal{P}}$ and $\tilde{D}$, while at the same time we pave the way for an algorithmic solution.

As discussed in Section 2, the general algorithmic approach to construct a connected graph in $\langle \mathcal{P}, D \rangle$ is to first construct a tree T in $[\mathcal{P}, D]$ and give T as input to Algorithm 1 to extend it to a connected graph in $\langle \mathcal{P}, D \rangle$. Recall that such a tree T satisfies three different constraints: connectivity, the upper bounds imposed by the edge constraints of D, and the upper bounds imposed by the degree constraints of d. As Lemma 3 shows, however, the latter set of constraints is the easiest to satisfy, and can be initially ignored.

A tree on V that does not violate the upper bounds imposed by the edge constraints of D, but may violate the upper bounds imposed by the degree constraints of d, will be called a *valid tree for $\mathcal{P}$ and D*. If such a tree exists, then Lemma 3 shows how to efficiently transform it into a tree in $[\mathcal{P}, D]$ and proceed as described above.

Lemma 3. *Let T_v be a valid tree for $\mathcal{P}$ and D. Then, we can efficiently construct a tree $T \in [\mathcal{P}, D]$.*

Proof. The proof is constructive and requires a polynomial number of steps. To construct T we modify the degrees of T_v within each V_i. Initially, $T = T_v$. Fix some $i \in [k]$, and let δ_i be the average degree of the vertices in V_i. Then, as long as $\max_{v \in V} \deg(v) > \lceil \delta_i \rceil$, we pick $x \in \arg\max_{v \in V} \deg(v)$ and $y \in \arg\min_{v \in V} \deg(v)$. Clearly $\deg(x) > \deg(y)$, so we can find a neighbor z of x such that the edge $yz \notin E(T)$. We then remove the edge xz from $E(T)$ and add the edge yz.

In each iteration a vertex x of large initial degree, i.e., $\deg(x) > \lceil \delta_i \rceil$, decreases its degree by one. So, in less than $\sum_{v \in V_i} \deg(v) = O(n^2)$ iterations all the degrees in V_i will be at most $\lceil \delta_i \rceil \leq d_i$. This is done for every i to get the desired T.

Notice that, while the degrees change, the number of edges between V_i and V_j is not affected, for any $i, j \in [k]$. Thus, $T \in [\mathcal{P}, D]$. □

Note that if $G \in \langle \mathcal{P}, D \rangle$ is connected, then any spanning tree T of G is a tree in $[\mathcal{P}, D]$. In particular, T is a valid tree for $\mathcal{P}$ and D. Thus, a *certificate of non existence of a valid tree* is also a certificate of non existence of a connected graph in $\langle \mathcal{P}, D \rangle$. In general, however, it is not clear how to construct efficiently a valid tree or a certificate of non existence of such a tree. Indeed, the sufficient and necessary condition for connectivity given in Subsection 3.2 appears to require exponential search. Nevertheless, our *Valid-Tree-Construct* algorithm (Algorithm 3) solves both problems in polynomial time.

Before we proceed to the technical details, we should note that there is a very specific type of valid trees we need to focus on. As the edge requirements may create a local shortage of edges, it is reasonable to think that trying to connect the vertices of V_i with each other as much as possible would only help. As a result, instead of any valid tree for $\mathcal{P}$ and D, we could try to produce a valid tree T such that the subgraph of T induced by V_i has the minimum possible number of connected components for every i. We are not going to prove this explicitly at this point, as it essentially is a corollary of Theorem 5, but it gives the intuition behind the following construct.

Given an instance $\mathcal{P}$, D, in order to look for a valid tree, we are going to consider a somewhat simpler instance. Consider a graph $G \in \langle \mathcal{P}, D \rangle$ such that for all $i \in [k]$ the subgraph of G induced by V_i has the minimum possible number of connected components. Now suppose that we contract the vertices of each such component into a single vertex and

delete loops and multiple edges to get a graph G'. It is not hard to show (see Lemma 4) that finding a spanning tree of G' is as good as finding a spanning tree of G. So, let $\tilde{\mathcal{P}} = \{\tilde{V}_1, \tilde{V}_2, \ldots, \tilde{V}_k\}$, where $|\tilde{V}_i| = \max\{1, |V_i| - d_{ii}\}$ is the minimum possible number of connected components of the subgraph induced by V_i in any realization in $\langle \mathcal{P}, D \rangle$. Moreover, let $\tilde{D} = (\tilde{d}_{ij})$ be derived from D in the natural way: $\tilde{d}_{ii} = 0$ and $\tilde{d}_{ij} = \min\{|\tilde{V}_i| \cdot |\tilde{V}_j|, d_{ij}\}$.

Lemma 4. *A valid tree $\tilde{T}$ for $\tilde{\mathcal{P}}$ and $\tilde{D}$ can be transformed efficiently into a valid tree T for $\mathcal{P}$ and D.*

Proof. To turn $\tilde{T}$ into a tree T on V that satisfies the d_{ij}s as upper bounds, we replace an arbitrary vertex of $\tilde{V}_i$ with a path of length $|V_i| - |\tilde{V}_i|$ so that we get $|V_i|$ vertices. We do this for every i. Now for the resulting tree T we have that the edges of T inside V_i are $|V_i| - |\tilde{V}_i|$. Recall that, by definition, $|\tilde{V}_i| = \max\{1, |V_i| - d_{ii}\}$, and thus the edges of T inside V_i are at most d_{ii}. Moreover, those paths inside each V_i were the only thing added to $\tilde{T}$. Therefore, the number of edges between V_i and V_j is at most $\tilde{d}_{ij} \leq d_{ij}$. That is, in time $O(n)$ we create a valid tree T for $\mathcal{P}$ and D. □

Lemmata 3 and 4 reduce the original algorithmic problem to finding a valid tree for $\tilde{\mathcal{P}}$ and $\tilde{D}$. Indeed, suppose that somehow we construct such a tree T'' (this is what Algorithm 3 in Subsection 3.3 does). Then, by Lemma 2, T'' can be efficiently transformed into a valid tree T' for $\mathcal{P}$ and D. Further, by Lemma 1, T' can be efficiently transformed into a tree $T \in [\mathcal{P}, D]$. Finally, T may be given as an initial seed to Algorithm 1 and a connected graph in $\langle \mathcal{P}, D \rangle$ is returned.

3.2 The Necessary and Sufficient Condition

We are almost ready to state the necessary and sufficient condition for the existence of a connected graph in $\langle \mathcal{P}, D \rangle$ in terms of the reduced instance $\tilde{\mathcal{P}}$, $\tilde{D}$.

In the simple second example mentioned at the beginning of the current section we would have $|\tilde{V}_1| = 6$, $|\tilde{V}_2| = 1$, $|\tilde{V}_3| = 2$, $|\tilde{V}_4| = 1$, $\tilde{d}_{13} = 6$, $\tilde{d}_{23} = \tilde{d}_{24} = 1$ and for all the other edge requirements with $i \leq j$, $\tilde{d}_{ij} = 0$. It is immediately apparent that there is no valid tree (or any tree whatsoever) since we have 10 vertices and only 8 edges. In general, however, things are more subtle. Having enough edges globally is still not sufficient. One has to connect vertices in $\tilde{V}_j$ using paths that go outside $\tilde{V}_j$ while respecting the local edge requirements. In fact, in order for the "local shortage of edges" to be revealed, it can be the case that several groups of $\tilde{V}_i$s need to be collapsed together, each in a single vertex. At this point we need to introduce some extra notation that serves this exact purpose. For every grouping of some of the $\tilde{V}_i$s we define a weighted graph. Each vertex of this graph is either some collapsed $\tilde{V}_i$ (the local requirements of which we want to highlight) or a collapsed group of $\tilde{V}_i$s (to which we may pay less attention). The weights are defined as to reflect the relevant edge requirements.

Let $\mathcal{F} \subseteq \tilde{\mathcal{P}}$, and let $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_\lambda\}$ be a partition of $\mathcal{F}$. As discussed above, the interpretation is that each $\mathcal{A}_i$ will collapse to a single vertex. The undirected weighted graph $\mathcal{G}_{\mathcal{F},\mathcal{A}} = (\mathcal{V}, \mathcal{E}, w)$ is defined as follows:

- $\mathcal{V} = \{\alpha_1, \alpha_2, \ldots \alpha_\lambda\} \cup \bigcup_{j:\tilde{V}_j \notin \mathcal{F}} \{u_j\}$, i.e., one vertex α_i for each $\mathcal{A}_i$ and one vertex u_j for each $\tilde{V}_j \notin \mathcal{F}$.

– For any $i, j \in [\lambda]$ with $i \neq j$, if there exist $\tilde{V}_x \in \mathcal{A}_i$ and $\tilde{V}_y \in \mathcal{A}_j$ such that $\tilde{d}_{xy} > 0$, then $\alpha_i\alpha_j \in \mathcal{E}$ and $w(\alpha_i\alpha_j) = 1$.

– For any $i \in [\lambda]$ and any $\tilde{V}_j \notin \mathcal{F}$, if there exists $\tilde{V}_x \in \mathcal{A}_i$ such that $\tilde{d}_{xj} > 0$, then $\alpha_i u_j \in \mathcal{E}$ and $w(\alpha_i u_j) = \min\left\{|\tilde{V}_j|, \sum_{x:\tilde{V}_x \in \mathcal{A}_i} \tilde{d}_{xj}\right\}$.

– For any $\tilde{V}_i \notin \mathcal{F}$, $\tilde{V}_j \notin \mathcal{F}$ with $i \neq j$, if $\tilde{d}_{ij} > 0$, then $u_i u_j \in \mathcal{E}$ and $w(u_i u_j) = \tilde{d}_{ij}$.

We can now state the necessary and sufficient condition for a connected graphic realization to exist.

Theorem 5. *There exists a connected $G \in \langle \mathcal{P}, D\rangle$ if and only if for every $\mathcal{F} \subseteq \tilde{\mathcal{P}}$ and every partition $\mathcal{A}$ of $\mathcal{F}$,*

$$\sum_{e\in\mathcal{E}} w(e) \geq |\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i| - 1$$

holds for the graph $\mathcal{G}_{\mathcal{F},\mathcal{A}} = (\mathcal{V}, \mathcal{E}, w)$ defined above.

The fact that the condition stated in Theorem 5 is necessary is relatively straightforward, and it is proved in Lemma 6 below. The challenging part is showing that these conditions are also sufficient, and this follows from the proof of Theorem 7 that deals with the correctness of Algorithm 3.

Lemma 6. *If there is a connected $G \in \langle \mathcal{P}, D\rangle$, then for every $\mathcal{F}$ and $\mathcal{A}$, $\sum_{e\in\mathcal{E}} w(e) \geq |\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i| - 1$ holds for $\mathcal{G}_{\mathcal{F},\mathcal{A}}$.*

Proof. Given a connected graph $G \in \langle \mathcal{P}, D\rangle$ we can easily get a connected graph $\tilde{G}$ on $\bigcup_{i=1}^{k} \tilde{V}_i$ that satisfies with equality the constraints imposed by $\tilde{D}$, i.e., there are exactly $\tilde{d}_{ij}$ edges between $\tilde{V}_i$ and $\tilde{V}_j$ for all $i, j \in [k]$. To do so, we contract each connected component of the subgraph G_i of G induced by V_i into a single vertex, and if necessary a few of these vertices together, so that the cardinality of the vertices of G_i reduces from $|V_i|$ to $|\tilde{V}_i|$. Then, we delete any loops or multiple edges. The resulting graph is still connected, since vertex contractions never increase the number of connected components. We do this for every $i \in [k]$ to get $\tilde{G}$. The edge constraints are satisfied by the definition of $\tilde{D}$.

Now for any $\mathcal{F} \subseteq \tilde{\mathcal{P}}$ and any partition $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_\lambda\}$ of $\mathcal{F}$, we may collapse each $\mathcal{A}_j$ in $\tilde{G}$ into a single vertex. To be precise, starting with $\tilde{G}$, for each $j \in [\lambda]$ we contract $\bigcup_{\tilde{V}_i \in \mathcal{A}_j} \tilde{V}_i$ into a single vertex α_j, and we delete any loops or multiple edges. The resulting graph G^* is still connected and has $|\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i|$ vertices. Therefore,

$$E(G^*) \geq |\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i| - 1\,. \tag{1}$$

Moreover, it is not hard to see that by contracting each $\tilde{V}_i$ left in G^* into a single vertex u_i, and then deleting any loops or multiple edges, we would get (the unweighted version of) $\mathcal{G}_{\mathcal{F},\mathcal{A}} = (\mathcal{V}, \mathcal{E}, w)$. To complete the proof, notice that between α_i and $\tilde{V}_j \notin \mathcal{F}$ there are $\min\left\{|\tilde{V}_j|, \sum_{x:\tilde{V}_x \in \mathcal{A}_i} \tilde{d}_{xj}\right\}$ edges in $E(G^*)$, which is 0 when $\alpha_i u_j \notin \mathcal{E}$ and $w(\alpha_i u_j)$ otherwise. Similarly, between $\tilde{V}_i \notin \mathcal{F}$ and $\tilde{V}_j \notin \mathcal{F}$ there are $\tilde{d}_{ij}$ edges in $E(G^*)$, which is 0 when $u_i u_j \notin \mathcal{E}$

and $w(u_iu_j)$ otherwise. Finally, $\alpha_i\alpha_j \in E(G^*)$ if and only if $w(\alpha_i, \alpha_j) = 1$ in $\mathcal{G}_{\mathcal{F},\mathcal{A}}$. Therefore,

$$E(G^*) = \sum_{e\in\mathcal{E}} w(e)\,. \tag{2}$$

Combining (1) and (2) gives $\sum_{e\in\mathcal{E}} w(e) \geq |\mathcal{A}| + \sum_{\tilde{V}_i\notin\mathcal{F}} |\tilde{V}_i| - 1$. □

3.3 Completing the Algorithmic Picture

Next, we state the polynomial time *Valid-Tree-Construct* algorithm (Algorithm 3) that either constructs a valid tree $\tilde{T}$ for $\tilde{\mathcal{P}}$ and $\tilde{D}$, or produces a certificate that no connected realization of $\langle \mathcal{P}, D\rangle$ exists, i.e., a pair $(\mathcal{F}, \mathcal{A})$ such that the above necessary condition fails.

The high level description of the algorithm is as follows. Let $\tilde{V} = \bigcup_{i=1}^{k} \tilde{V}_i$. The algorithm tries to construct a tree on $\tilde{V}$ by maintaining exactly $|\tilde{V}| - 1$ edges which are valid for $\tilde{\mathcal{P}}$ and $\tilde{D}$ (i.e., the number of edges between each $\tilde{V}_i$ and $\tilde{V}_j$ never exceeds $\tilde{d}_{ij}$), while at the same time decreasing the number of connected components by adding and removing edges appropriately. The main idea is that if two components cannot be connected in a trivial way that maintains validity, then the $\tilde{V}_i$s that intersect more than one connected component play a critical role. We constantly try to "free" an edge incident to such a $\tilde{V}_i$ while preserving validity and not increasing the number of connected components. In the case that such a $\tilde{V}_i$ intersects a cycle, this is an easy task. Otherwise, we have to remove all the $\tilde{V}_i$s that intersect more than one component, and try to connect two components in the resulting graph. We recursively repeat this until we connect something, and then it is easy to find a sequence of adding and removing edges that connects two components in the original graph and maintains validity. If the recursion fails, we have a certificate that no connected realization exists.

Before we proceed with the statement and the analysis of Algorithm 3, we need to clarify how we construct an initial graph G_0 (line 1 of the algorithm) on $|\tilde{V}|$ vertices and $|\tilde{V}|-1$ edges, that does not violate any upper bounds imposed by $\tilde{D}$. This can be achieved by running JDM-Construct on an appropriate input for a restricted number of steps. Specifically, if $G_\emptyset$ is the empty graph on $\tilde{V}$, and δ is a $|\tilde{V}|$-dimensional vector with every coordinate equal to a sufficiently large number, e.g., $|V|$, then running JDM-Construct$(G_\emptyset, \tilde{V}, \delta, \tilde{D})$ for $|\tilde{V}| - 1$ iterations will produce such a graph G_0.

To facilitate the presentation we abuse the notation and write things like $G_j \leftarrow G \cup P_j \cup Z_j$ instead of $G_j \leftarrow \big(V(G) \cup \bigcup_{\tilde{V}_i\in P_j} \tilde{V}_i\,,\, E(G) \cup Z_j\big)$, or $G_{j+1} \leftarrow G_j \setminus P_j$ instead of saying that G_{j+1} is the subgraph of G_j induced by $V(G_j) \setminus \bigcup_{\tilde{V}_i\in P_j} \tilde{V}_i$.

Theorem 7. *Algorithm 3 outputs either a valid tree for $\tilde{\mathcal{P}}$ and $\tilde{D}$, or a certificate $(\mathcal{F}, \mathcal{A})$ showing that no such tree exists. The algorithm runs in time polynomial in n.*

Proof. The algorithm starts with a graph G_0 on $|\tilde{V}|$ vertices and $|\tilde{V}| - 1$ edges, that does not violate any upper bounds imposed by $\tilde{D}$. Then, if G_0 is not connected, the algorithm either connects two connected components $\mathcal{C}, \mathcal{C}'$ of G_0 to each other without violating any $\tilde{d}_{ij}$ (Cases 1 and 2, lines 9-11 and 12-16 respectively), or outputs a certificate of non existence of a valid tree for $\tilde{\mathcal{P}}$ and $\tilde{D}$ (Case 3, lines 17-21), or creates an induced subgraph G_1 of G_0 (Case 4, lines 22-23) and then attempts to do the above for G_1.

At first glance, it is not obvious that the number of graphs $G_1, G_2, \ldots$ created this way is bounded, or that the value of j itself (the depth of the recursion) does not change arbitrarily

Algorithm 3: Valid-Tree-Construct$\left(\tilde{\mathcal{P}}, \tilde{D}\right)$

Use JDM-Construct to produce a valid graph G_0 for $\left(\tilde{\mathcal{P}}, \tilde{D}\right)$ with $|\tilde{V}| - 1$ edges, where $\tilde{V} = \bigcup_{i=1}^{k} \tilde{V}_i$
$j \leftarrow 0$ /* the depth of the current recursion */
$G \leftarrow G_0$ /* G is an auxiliary graph */
while G_0 *is not connected* **do**
  $A_j \leftarrow \{v \mid v$ lies in some cycle of $G_j\}$
  $C_j \leftarrow \left\{\tilde{V}_i \mid A_j \cap \tilde{V}_i \neq \emptyset\right\}$
  $P_j \leftarrow \left\{\tilde{V}_i \mid \tilde{V}_i$ intersects at least two connected components of $G_j\right\}$
  $Z_j \leftarrow \left\{e \in G_j \mid$ at least one endpoint of e is in some $\tilde{V}_i \in P_j\right\}$
  if *there exists an edge e connecting two connected components of G without violating any upper bounds of $\tilde{D}$* **then** /* Case 1 */
    Add e to $E(G)$ and remove an edge that belongs to a cycle in G
    $j \leftarrow \max\{j-1, 0\}$; $G_j \leftarrow G \cup P_j \cup Z_j$; $G \leftarrow G_j$
  else if $C_j \cap P_j \neq \emptyset$ **then** /* Case 2 */
    Pick u, v in some $\tilde{V}_i \in C_j \cap P_j$ that belong in different connected components of G and $u \in A_j$
    Find a neighbor x of u that lies on the same cycle in G
    Remove xu from $E(G)$ and add $e = xv$
    $j \leftarrow \max\{j-1, 0\}$; $G_j \leftarrow G \cup P_j \cup Z_j$; $G \leftarrow G_j$
  else if $P_j = \emptyset$ **then** /* Case 3 */
    Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\xi$ be the connected components of G_j
    $\mathcal{A}_i \leftarrow \left\{\tilde{V}_x \mid \tilde{V}_x \subseteq V(\mathcal{C}_i)\right\}$ for $1 \leq i \leq \xi$
    $\mathcal{F} \leftarrow \bigcup_{i=1}^{\xi} \mathcal{A}_i$; $\mathcal{A} \leftarrow \{\mathcal{A}_1, \ldots, \mathcal{A}_\xi\}$
    return $(\mathcal{F}, \mathcal{A})$ /* found a certificate of non existence */
  else /* Case 4 */
    $G_{j+1} \leftarrow G_j \setminus P_j$; $G \leftarrow G_{j+1}$; $j \leftarrow j+1$
return G_0

before anything meaningful happens for G_0. Next we show that the algorithm works in phases. The ℓth phase consists of $j_\ell \geq 0$ iterations where Case 4 happens, and then either one iteration where Case 3 happens, or $j_\ell + 1$ iterations where Cases 1 and 2 happen.

Claim 1. Once Case 1 or Case 2 happen, say while $G = G_{j_\ell}$, then in the next j_ℓ iterations Cases 1 and 2 will happen.

Proof of Claim 1. For notational convenience, during the second half of the current phase we are going to use G'_j, G''_j for the jth level graph at the beginning and at the end respectively of the corresponding iteration, as opposed to G_j that denotes the "old" jth level graph during the first half of the phase. That is, for $j \leq j_\ell$, G_j is the graph created at the end of the jth iteration of the current phase, while G'_j and G''_j denote the corresponding graph at the

beginning and at the end of the $(2j_\ell - j)$th iteration.

We are going to prove the statement by induction on j_ℓ. In fact, we are going to augment it with the extra statement "Moreover, the vertex sets of the connected components of G'_0 and G_0 are the same."

When $j_\ell = 0$ the statement trivially holds, so assume $j_\ell > 0$. Note that if we think of G_1 as the starting point of the recursion, the number of iterations before Cases 1 or 2 happen is $j_\ell - 1$. Thus, by the inductive hypothesis, once Case 1 or Case 2 happen when $G = G_{j_\ell}$, then in the next $j_\ell - 1$ iterations Cases 1 and 2 will happen. Moreover, the vertex sets of the connected components of G'_1 and G_1 are the same.

Since one of Cases 1 or 2 happens in the very last iteration that is implied by the inductive hypothesis, two connected components of $G = G'_1$ become connected to each other without any $\tilde{d}_{ij}$ being violated. Let e, e' denote the edges added and removed respectively from G'_1 during this iteration, i.e., $G''_1 = (G'_1 - e') + e$. At the end of the iteration, G'_0 is created by adding P_0 and Z_0 to G''_1, and it becomes the current graph G.

Since e' belonged in a cycle, G'_1 and $G''_1 - e$ have the same connected components. Thus, the vertex sets of the connected components of G_1 and $G''_1 - e$ are the same. By adding P_0 and Z_0 to both G_1 and $G''_1 - e$, we directly get that the vertex sets of the connected components of G_0 and $G'_0 - e$ are the same. Next, toward a contradiction, suppose that G'_0 has less connected components than $G'_0 - e$. It follows that e connects two connected components in $G'_0 - e$, and consequently e connects the corresponding connected components in G_0. But then in the very first iteration of the current phase Case 1 happens, and thus $j_\ell = 0$. This contradicts the fact that $j_\ell > 0$. Therefore, G'_0 has the same connected components as $G'_0 - e$, and we conclude that the vertex sets of the connected components of G_0 and G'_0 are the same.

The latter result implies that P_0 is not affected, i.e., $P'_0 = P_0$. Moreover, the above discussion implies that the endpoints of e, that are not connected in $G''_1 - e$, are connected in $G'_0 - e$. The key observation now is that a new cycle is created in G'_0, containing the new edge e, as well as some v from some $\tilde{V}_i \in P_0$ that was on a path connecting the endpoints of e in $G'_0 - e$. That is, in the next iteration, if Case 1 does not happen, Case 2 does (since for G'_0 we have $C_0 \cap P_0 \supseteq \{\tilde{V}_i\}$). This completes the inductive step. ⊡

Claim 1 implies that once Case 1 or Case 2 happen, then after j_ℓ more iterations the number of connected components of G_0 is decreased by one, without any $\tilde{d}_{ij}$ being violated. Clearly, the number of phases is upper bounded by $|\tilde{V}|$. Moreover, we bound each j_ℓ by k, i.e., the number of degree classes.

Claim 2. Within k iterations one of Cases 1, 2 or 3 happens.

Proof of Claim 2. Notice that when G_0 is not a tree, it contains a cycle, i.e., $A_0 \neq \emptyset$. Moreover, whenever G_j is created in Case 4, its vertex set contains A_{j-1}, or the current iteration would not go further than Case 2. That is, G_j—if created at all—contains all the cycles of G_0. This also implies that whenever G_j is created, $V(G_j) \neq \emptyset$.

On the other hand, G_j is the subgraph of G_{j-1} induced by $V(G_{j-1}) \setminus \bigcup_{\tilde{V}_i \in P_{j-1}} \tilde{V}_i$. But G_j is constructed in Case 4, and thus $P_{j-1} \neq \emptyset$, or the algorithm would have terminated in Case 3. Therefore, $V(G_{j-1}) \setminus V(G_j)$ contains one or more of the $\tilde{V}_i$s. As discussed above, whenever G_j is created, $V(G_j) \neq \emptyset$, so the algorithm cannot go to Case 4 for k consecutive iterations. That is, within k iterations one of Cases 1, 2, or 3 happens. ⊡

Combining Claims 1 and 2, we get that within no more than $2k+1$ iterations Algorithm 3 either reduces the number of components of G_0, or terminates in Case 3. So, in at most $(2k+1)|\tilde{V}|$ iterations the algorithm terminates and, if Case 3 never happens, the graph G_0 returned is a valid tree for $\tilde{\mathcal{P}}$ and $\tilde{D}$. Notice that every iteration takes time $O(|\tilde{V}|^2)$, hence the running time of the algorithm is $O(n^4)$.

Now suppose that Case 3 happens. That is, for some j, $P_j = \emptyset$ and the conditions in lines 9 and 12 fail. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\xi$ be the connected components of G_j and let $\mathcal{A}$ and $\mathcal{F}$ be defined as in lines 19-20 of the algorithm. Note that the definition of the $\mathcal{A}_i$s (and thus of $\mathcal{A}$ and $\mathcal{F}$ as well) makes sense. Indeed, from the way G_j is constructed, if a vertex $v \in \tilde{V}_x$ is in $V(G_j)$, then $\tilde{V}_x \subseteq V(G_j)$. Therefore, since $P_j = \emptyset$, if a vertex $v \in \tilde{V}_x$ is in a component $\mathcal{C}_i$, then $\tilde{V}_x \subseteq V(\mathcal{C}_i)$.

Consider the graph G_j together with all vertices and edges removed in the previous j iterations, i.e., the current G_0.

Claim 3. In the current G_0, any edge that can be added without violating the upper bounds given by $\tilde{D}$, must have both its endpoints in some $\mathcal{C}_i$.

Proof of Claim 3. It suffices to show that all other possible cases fail to happen. Suppose that there exists an edge uv that can be added to G_0 without violating any constraint, such that $u \in V(\mathcal{C}_i)$ and $v \in V(\mathcal{C}_{i'})$, where $i \neq i'$. But then, uv would have been found and added in Case 1 of the current iteration. Now, suppose that there exists an edge uv that can be added to G_0 without violating any constraint, such that either $u \in V(\mathcal{C}_i), v \in \tilde{V}_y \in P_{i'}$ with $i' < j$, or $u \in \tilde{V}_x \in P_i, v \in \tilde{V}_y \in P_{i'}$ with $i' \leq i < j$. However, uv was also available a few iterations back, when G was $G_{i'}$. Since $\tilde{V}_y$ intersects at least two connected components of $G_{i'}$, there exists $v' \in \tilde{V}_y$ such that u and v' lie in different connected components in $G_{i'}$. Because adding uv is legal, so is adding uv'. Hence, uv' would have been added in Case 1 of that iteration. We conclude that the claim is true. ⊡

By Claim 3, for any $\tilde{V}_x, \tilde{V}_y$ that are not contained in $\mathcal{F}$, we must have as many edges as possible between $\tilde{V}_x$ and $\tilde{V}_y$ in G_0. That is, if $G[X,Y]$ denotes the bipartite subgraph of G induced by X and Y,

$$\left|E\big(G_0\big[\tilde{V}_x, \tilde{V}_y\big]\big)\right| = \tilde{d}_{xy}\,.$$

Similarly, for any $\tilde{V}_x \notin \mathcal{F}$ and any $\mathcal{A}_i$ we must already have as many edges as possible in G_0 between $\tilde{V}_x$ and all the $\tilde{V}_y$s in $\mathcal{A}_i$. That is,

$$\left|E\left(G_0\left[\tilde{V}_x\,,\ \bigcup_{\tilde{V}_y \in \mathcal{A}_i} \tilde{V}_y\right]\right)\right| = \sum_{y:\tilde{V}_y \in \mathcal{A}_i} \tilde{d}_{xy}\,.$$

Now we contract each $\mathcal{A}_i$ into a single vertex and delete loops and multiple edges to get H from G_0. For the number of vertices and edges of H it is easy to see that

$$|V(H)| = |\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i|\,, \tag{3}$$

and

$$|E(H)| = \frac{1}{2} \sum_{x:\tilde{V}_x \notin \mathcal{F}} \sum_{y:\tilde{V}_y \notin \mathcal{F}} \tilde{d}_{xy} + \sum_{\tilde{V}_x \notin \mathcal{F}} \sum_{i=1}^{\xi} \min\left\{|\tilde{V}_x|,\ \sum_{y:\tilde{V}_y \in \mathcal{A}_i} \tilde{d}_{xy}\right\}. \tag{4}$$

The next step is to relate $|V(H)|$ and $|E(H)|$ with each other.

Claim 4. $|E(H)| < |V(H)| - 1$.

Proof of Claim 4. It suffices to show that H remains disconnected, but contains no cycles. The former is straightforward, since G_0 is disconnected, and contracting sets of vertices that induce connected subgraphs (like $\bigcup_{\tilde{V}_j \in \mathcal{A}_i} \tilde{V}_j$) does not make it connected. To see that H is acyclic, we need the observation made in the first part of the proof of Claim 2: G_j contains all the cycles of G_0. Thus, by contracting the sets of vertices that induce the connected components of G_j—which is what we do by contracting each $\mathcal{A}_i$ to get H—we get rid of all the cycles of G_0. ⊡

It is straightforward to see that if we use $\mathcal{F}$ and $\mathcal{A}$ to construct the weighted graph $\mathcal{G} = (\mathcal{P}, \mathcal{E}, w)$ as in the necessary and sufficient condition, then $\sum_{e \in \mathcal{E}} w(e) = |E(H)|$. If we combine this with Claim 4 and inequalities (3) and (4), we have

$$\sum_{e \in \mathcal{E}} w(e) < |\mathcal{A}| + \sum_{\tilde{V}_i \notin \mathcal{F}} |\tilde{V}_i| - 1 \,.$$

That is, the condition fails to hold and $(\mathcal{F}, \mathcal{A})$ is indeed a certificate showing that no connected graph in $\langle \mathcal{P}, D \rangle$ exists. □

Note that the correctness of Algorithm 3 completes the proof of Theorem 5 as well. Below (Algorithm 4) we give a high level description of all the steps involved in the construction of a connected realization in $\langle \mathcal{P}, D \rangle$. The running time of Valid-Tree-Construct, i.e., $O(n^4)$, dominates the running time of Algorithm 4.

Algorithm 4: JDM-Connected-Construct$(\mathcal{P}, D)$

```
Create P̃ and D̃                                   /* as described before Lemma 4 */
Run Valid-Tree-Construct(P̃, D̃)                                  /* Algorithm 3 */
if Valid-Tree-Construct outputs (F, A) then
    return (F, A)                  /* there is no connected graph in ⟨P, D⟩ */
else
    Let T'' be the output Valid-Tree-Construct(P̃, D̃)
    Using T'', construct a valid tree T' for P and D                 /* Lemma 4 */
    Using T', construct a tree T ∈ [P, D]                            /* Lemma 3 */
    return JDM-Construct(T, P, d, D)          /* Connected by Theorem 2 */
```

Acknowledgments

G. Amanatidis was supported in part by an ACO and an ARC Georgia Tech Fellowship, and NSF-CCF-TF-0830683. B. Green was supported in part by an NSF VIGRE Fellowship. M. Mihail was supported in part by NSF-CCF-0539972 and NSF-CCF-TF-0830683. The authors would like to thank the anonymous referee for their helpful and constructive comments.

References

[1] P. Erdős, T. Gallai, Graphs with prescribed degrees of vertices, Math. Lapok 11.

[2] C. Berge, Graphs and Hypergraphs, Elsevier Science Ltd., Oxford, UK, 1985.

[3] V. Havel, A remark on the existence of finite graphs, Kaposis Pest Mat 80.

[4] S. Hakimi, On the realizability of a set of integers as degrees of the vertices of a graph, SIAM, J. Applied Math. 10.

[5] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, k. claffy, A. Vahdat, The internet as-topology: Three data sources and a definitive metric, ACM Sigcomm CCR.

[6] P. Mahadevan, D. Krioukov, K. Fall, A. Vahdat, Systematic topology analysis and generation using degree correlations, Sigcomm.

[7] M. Gjoka, B. Tillman, A. Markopoulou, Construction of simple graphs with a target joint degree matrix and beyond, in: 2015 IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1553–1561.

[8] G. Amanatidis, B. Green, M. Mihail, Graphic realizations of joint-degree matrices, Unpublished Manuscript CoRR, abs/1509.07076.
URL `http://arxiv.org/abs/1509.07076`

[9] I. Stanton, A. Pinar, Constructing and sampling graphs with a prescribed joint degree distribution, J. Exp. Algorithmics 17 (2012) 3.5:3.1–3.5:3.25.

[10] E. Czabarka, A. Dutle, P. L. Erdős, I. Miklós, On realizations of a joint degree matrix, Discrete Applied Mathematics 181 (2015) 283 – 288.

[11] A. Patrinos, S. Hakimi, Relations between graphs and integer-pair sequences, Discrete Mathematics 15 (4) (1976) 347–358.