Better Trees for Santa Claus

Étienne Bamas
etienne.bamas@epfl.ch
EPFL
Lausanne, Switzerland

Lars Rohwedder
l.rohwedder@maastrichtuniversity.nl
Maastricht University
Maastricht, Netherlands

ABSTRACT
We revisit the problem max-min degree arborescence, which was introduced by Bateni et al. [STOC'09] as a central special case of the general Santa Claus problem, which constitutes a notorious open question in approximation algorithms. In the former problem we are given a directed graph with sources and sinks and our goal is to find vertex disjoint arborescences rooted in the sources such that at each non-sink vertex of an arborescence the out-degree is at least $k$, where $k$ is to be maximized.

This problem is of particular interest, since it appears to capture much of the difficulty of the Santa Claus problem: (1) like in the Santa Claus problem the configuration LP has a large integrality gap in this case and (2) previous progress by Bateni et al. was quickly generalized to the Santa Claus problem (Chakrabarty et al. [FOCS'09]). These results remain the state-of-the-art both for the Santa Claus problem and for max-min degree arborescence and they yield a polylogarithmic approximation in quasi-polynomial time. We present an exponential improvement to this, a poly$(\log \log n)$-approximation in quasi-polynomial time for the max-min degree arborescence problem. To the best of our knowledge, this is the main technical novelty of our result are locally good solutions: informally, we show that it suffices to find a poly$(\log n)$-approximation that locally has stronger guarantees. We use a lift-and-project type of LP and randomized rounding, which were also used by Bateni et al., but unlike previous work we integrate careful pruning steps in the rounding. In the proof we extensively apply Lovász Local Lemma and a local search technique, both of which were previously only used in the context of the configuration LP.

CCS CONCEPTS
• Theory of computation → Approximation algorithms analysis.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

STOC '23, June 20–23, 2023, Orlando, FL, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9913-5/23/06...$15.00
https://doi.org/10.1145/3564246.3585174

KEYWORDS
max-min, allocation, fairness, Lovász Local Lemma

ACM Reference Format:
Étienne Bamas and Lars Rohwedder. 2023. Better Trees for Santa Claus. In Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC ‘23), June 20–23, 2023, Orlando, FL, USA. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585174

1 INTRODUCTION
In the Santa Claus problem (also known as max-min fair allocation), there are gifts that need to be assigned to children. Each gift $j$ has unrelated values $v_{ij}$ for each of the children $i$. The goal is to assign each gift $j$ to a child $\sigma(j)$ such that we maximize the utility of the least happy child, that is, $\min_i \sum_j \sigma(j) \cdot v_{ij}$. The dual of the problem, where one has to minimize the maximum instead of maximizing the minimum is the problem of makespan minimization on unrelated parallel machines. Both variants form notoriously difficult open problems in approximation algorithms [5, 28, 31] and there is a common belief that the Santa Claus problem admits a constant approximation if and only if makespan minimization on unrelated machines admits a better-than-2 approximation [3]. Although formally no such reduction is known, techniques often seem to transfer from one problem to the other, see for example the restricted assignment case in the related work section.

Bateni, Charikar, and Guruswami [7] identified as a central special case of the Santa Claus problem the restriction that for all values we have $v_{ij} \in \{0, 1, \infty\}$ and for each gift there is at most one child with $v_{ij} = \infty$ and for each child there is at most one gift with $v_{ij} = \infty$. This case can be rephrased as the following graph problem, dubbed the max-min degree arborescence problem. We are given a directed graph $G = (V, E)$, a set of sources $S \subseteq V$, and a set of sinks $T \subseteq V$. Our goal is to compute a set of vertex disjoint arborescences rooted in each of the sources $S$. The leaves of these arborescences are in the sinks in $T$. For all inner vertices we have an out-degree of $k$, where $k$ is to be maximized. Bateni et al. gave a max{(poly$(\log n)$, $n^{\epsilon}$)}-approximation in time $2^{\tilde{O}(1/\epsilon)}$, which is still the state-of-the-art for this problem. In particular, they obtain a quasi-polynomial time polylogarithmic approximation by setting $\epsilon = \log \log n / \log n$. In a highly non-trivial way, the same approach was then generalized by Chakrabarty, Chuzhoy, and Khanna [9] to obtain the same result also for the Santa Claus problem. The only hardness known for the Santa Claus problem is that there is no better-than-2 approximation (see [8, 25]) and it is not difficult to show there is no better-than-$\sqrt{e}/(e-1)$ approximation for the max-min degree arborescence problem via a reduction from max-$k$-cover (see full version of the paper). This still leaves a large gap in the understanding of both problems. In particular, the two most pressing questions are whether the polylogarithmic guarantee can...
also be achieved in polynomial time and whether a sublogarithmic approximation guarantee can be achieved. In this paper we answer the latter question affirmatively for the max-min degree arborescence problem by giving a poly(log log n)-approximation in quasi-polynomial time.

While much progress on sublogarithmic approximations has been made on other special cases of the Santa Claus problem, most notable the restricted assignment case, these results are all based on the configuration LP. On the other hand, it is known that for the general Santa Claus problem the configuration LP has an unbounded integrality gap and hence these methods seem unlikely to generalize. Similarly, the max-min degree arborescence problem is a case where the integrality gap of the configuration LP is already high [7] and as such the algorithmic techniques need to be rethought. Given this and the fact that previous progress on the problem was quickly extended to the general Santa Claus problem we believe that the techniques introduced in this paper are highly relevant towards the goal of understanding the approximability of the Santa Claus problem.

### 1.1 Other Related Work

An influential line of work on the Santa Claus problem addresses the so-called restricted assignment case, which is the case where \( a_{ij} \in \{0, 1\} \) for any child \( i \) and gift \( j \). Here, \( a_{ij} \) is a uniform value of the gift. One may also rephrase this as the setting where all gifts have the same value for each child, but they cannot be assigned to all of them. In a seminal work, Bansal and Srividenko [6] provide a \( O(\log \log(n)/\log \log \log(n)) \)-approximation algorithm for this case using randomized rounding of the configuration LP using Lovász Local Lemma. This was improved by Feige [17] to an \( O(1) \)-approximation. Further progress on the constant or the running time was made since then, see e.g. [2, 3, 12, 13, 15, 20, 27], or extended to more general valuation functions [4]. Many of these results are based on a local search technique due to Haxell [21], who developed it in the context of hypergraph matching. The technique was later also adapted to the dual problem (makespan minimization) by Svensson [29] to give a better-than-2 approximation in the restricted assignment case of it as well. This result also led to a series of improvements either in the running time or the approximation factor [1, 10, 22, 24]. Unfortunately, all these results rely on the configuration LP; hence they seem unlikely to extend to the max-min degree arborescence problem or even the general Santa Claus problem. Perhaps surprisingly, the local search technique still plays a secondary, but important role in our proof, where we use it without the configuration LP. Also Lovász Local Lemma is important in our proof, although our use differs significantly from that of Bansal and Srividenko.

The Graph Balancing version of the Santa Claus problem is the special case where every gift has a non-zero value only for two children. Although this variant also has a graph structure, it is very different in nature from max-min degree arborescence. This problem is resolved with a remarkably clean polynomial time 2-approximation [30] and a matching lower bound. It is quite surprising that this works even when the values for the two children are unrelated, because the corresponding min-max version with unrelated values has not seen any progress so far (for the min-max version where the value is the same for both children see [16, 23]). Nevertheless, also the Graph Balancing special case of the Santa Claus problem has the striking feature that the configuration LP is strong, see for example [9].

Another problem that is at least intuitively related to the max-min arborescence is the directed Steiner Tree problem. In this problem we are given an edge-weighted directed graph with a source and a set of sinks. The goal is to find an arborescence of minimal weight, which is rooted at the source and spans all sinks. It is quite remarkable that the state-of-the-art for this problem is very similar to ours: there is a \( n^\epsilon \)-approximation algorithm in polynomial time for every fixed \( \epsilon > 0 \) and a polylogarithmic approximation algorithm in quasi-polynomial time [11]. Unlike our problem, it was shown that no sublogarithmic (in fact, no \( \log^{2-\epsilon} n \) approximation exists [19]. It may therefore come as a surprise that we can indeed find a sublogarithmic approximation for our problem. Related to the directed Steiner Tree problem is also the (undirected) group Steiner Tree problem, which can be shown to be a special case. In this problem, we are given an undirected weighted graph and a list of groups that are subsets of vertices. The goal is to compute the cheapest set of edges that is connected and contains at least one vertex from each group. Here, there are more subtle connections to our problem: taking a closer look at the literature one can notice that the challenging instances in the group Steiner Tree problem have a similar structure to the challenging ones for our case. More precisely they are layered graphs with \( O(\log n) \) layers. Halperin et al. [18] show that the integrality gap of a natural LP relaxation of group Steiner Tree could be amplified from \( \Theta(\log(n)) \) on \( O(1) \)-layered instances to \( \Theta(\log^2(n)) \) on \( \Omega(\log(n)) \)-layered graphs. This construction was later transformed to the hardness result by Halperin and Krauthgamer [19].

Before our work, it was quite plausible that such an amplification technique could also apply in the context of the max-min degree arborescence problem. This would have shown how to amplify a \( \Omega(1) \) gap on \( O(1) \)-layered instances to a \( \Theta(\log(n)) \) gap on \( \Theta(\log(n)) \)-layered instances. In the full version of the paper, we adapt the construction of [18] to our setting. At first sight, the gap indeed seems to amplify and our construction shows that the previous rounding algorithms of [7, 9] cannot hope to get better than a \( \Omega(\log(n)) \)-approximation. Fortunately, we notice that a single round of top-to-bottom pruning (throwing away half the children of every vertex) seems to resolve that issue. We note that the group Steiner Tree problem has a rich history and it would be very interesting to see if more techniques could be transferred to the max-min degree arborescence problem.

### 1.2 Informal Overview of Techniques

A crucial idea that goes back to previous work [7, 9] is to allow congestion in the solution. Generally, a vertex can only have one incoming edge in the solution, but we relax this constraint. We call the maximum number of times a vertex is used the congestion. The algorithms in [7, 9] employ randomized rounding to obtain a solution with polylogarithmic congestion. The congestion can then be translated into an approximation rate by relatively straightforward arguments. This polylogarithmic congestion comes from...
a Chernoff bound that yields an inversely polynomial probability, which is then applied to all vertices with a union bound.

A new ingredient of our algorithm is the notion of local congestion. Roughly speaking, we first compute a solution, which still has polylogarithmic congestion, but when considering only a local part of the solution (say, vertices within a distance of \( \ell = O(\log \log n) \) in the arborescences), then this local part needs to have much smaller congestion, i.e., poly(log log n). In other words, if a vertex is used multiple times in the solution, then the occurrences should be far apart in the arborescences.

First, let us describe why it is plausible to be able to obtain such a guarantee. It is already known that by randomized rounding a polylogarithmic congestion can be achieved. The local congestion on the other hand is by definition a very local constraint and hence Lovász Local Lemma (LLL) is natural to employ. This is indeed our approach, although the details are challenging.

Next, we will explain how to arrive at a sublogarithmic congestion. The approach that we call top-to-bottom pruning is very blunt: slightly oversimplifying, we take a given solution (with poly(log n) congestion) and start at the sources of the arborescences. We throw away randomly a constant fraction (say, half) of their children. Then we move to the other children and recurse. Clearly, this decreases the approximation rate only insignificantly. However, in expectation the congestion at each vertex decreases drastically. If for example a vertex is at distance \( d \) from the source of an arborescence, then the probability of it not being removed is only \( 1/2^d \).

There is, however, a caveat here: suppose that the same vertex occurs in a (relaxed) arborescence many times and all occurrences are very close to each other. Then there is a high positive correlation between the vertices not being removed, which forms a serious problem. Indeed, this is where the local congestion comes in. It essentially bounds the dependence of occurrences surviving. Again, this proof makes use of LLL, because it seems infeasible to try and make the probability of a vertex’s congestion staying above poly(log log n) small enough to apply a union bound.

Since both parts require LLL, it is crucial to bound the dependencies. However, if \( k \) is large (say \( \Omega(n) \)), then even a local part of the arborescence contains many vertices. It then seems unlikely to be able to guarantee locally low congestion for every vertex. Roughly speaking, we will only guarantee the property for a large fraction of the vertices, so that we can make the probability inversely polynomial in \( k \). All other vertices need to be removed from the solution and this is generally very dangerous: even if we remove only a small fraction of vertices, this can lead to other vertex removals becoming necessary, because they now have a low out-degree. If we are not careful, this can accelerate and corrupt the whole solution. We call this the bottom-to-top pruning and we formalize a condition, under which the damage to the solution can be controlled. This condition is then applied in both parts.

This brings us to a discussion on our two pruning techniques. Intuitively, both approaches have complementary merits to each other. Bottom-to-top pruning makes it easy to maintain a low maximum congestion, but difficult to keep a good number of children for every vertex in the arborescence. On the contrary, top-to-bottom pruning makes it very easy to maintain a good number of children, but difficult to keep the maximum congestion under control. Our proof can be seen as a careful combination of those two techniques using LLL. The use of pruning makes the proof fairly involved and one might wonder if this could not be avoided. In particular, it is not clear if the analysis of the previous randomized rounding algorithm (see [7, 9]) is tight or not. However, we argue that an \( \Omega(\log n/\log \log n) \) factor seems unavoidable in previous works, and that it is not easily fixable. We now elaborate on this: In previous approaches as in ours, a crucial part of the algorithm is to solve the max-min degree arborescence problem on layered instances (that is, the sources are located in the first layer, and edges can exist only between vertices of consecutive layers). Previous works [7, 9] then solve these instances by rounding an LP relaxation of the problem (we use the same LP relaxation but with a different rounding). An intuitive randomized rounding that appears in previous works is roughly as follows. Assume that the LP says there exists a solution of value \( k \). Then the source samples \( k \) children with probability equal to the LP values. These selected children then select \( k \) children each, again equal to the LP values (more precisely, values that correspond to conditioning on the previous selections). In that manner, we make progress layer by layer until reaching the last layer. This guarantees a maximum congestion that is at most polylogarithmic in the number of vertices, hence the polylogarithmic approximation ratio. At this point, one might be tempted to argue that very few vertices in our solution will have congestion \( \Omega(\log n) \) and that they are not a serious problem. Unfortunately, we show an instance in which the above rounding results in a solution in which all the sinks selected in the arborescence have an expected congestion of \( \Omega(\log n/\log \log n) \) (see full version of the paper). While this may seem counter-intuitive, recall that here we are implicitly conditioning by the fact of being selected in the solution. It is non-trivial to recover from this issue: For instance deleting—in a bottom-to-top fashion—the vertices with high congestion will basically remove almost all the sinks which will corrupt the whole solution. In fact, we argue that the bottom-to-top pruning is very sensitive to a small number of deletions if one aims at a sublogarithmic approximation ratio (see full version of the paper). We believe this is another reason why getting such a guarantee is challenging.

Lastly, there is one important issue that we have not mentioned so far: in a similar way that each vertex loses some fraction of its children in the rounding (compared to the LP relaxation), we would lose some fraction of the sources. This happens also in previous works [7, 9], who then only compute a solution for a \( 1/poly(\log n) \) fraction of the sources and repeat it for \( poly(\log n) \) times to cover all sources. This again introduces a congestion of poly(log n), which seems difficult to avoid with the randomized rounding approach. First, we only design the randomized rounding to approximate single source instances so that we do not have to cope with this. Then we present a black-box reduction from many sources to one source, which uses a non-trivial machinery that is very different from the randomized rounding approach. Namely, this is by a local search framework, which has already seen a big application in the restricted assignment case of the Santa Claus problem, see related work. Usually the local search is analyzed against the configuration LP, which is not applicable here, so our way of applying it is quite different to previous works.
1.3 Relevance to the Santa Claus Problem
While it may seem as if the techniques introduced in this paper are limited to the max-min degree arborescence problem, we believe that they are of interest also towards obtaining a better approximation for the Santa Claus problem. We emphasize that in the state-of-the-art [9], the authors arrive through a series of sophisticated reductions at a similar problem as the layered instances of max-min bounded arborescence (see next section for definition of layered instances). The main difference is that they cannot simply select k outgoing edges for each vertex independently. Instead, the k children have to be connected to the parent through a capacitated flow network shared by all vertices. For details we refer to [9].

An obstacle in adapting our arguments to that setting is that these flow networks introduce new dependencies, which seem difficult to bound and our LL arguments are naturally very sensitive to dependencies. On the other hand, the general idea of using locally low congestion solutions as an intermediate goal and applying methods similar to the top-to-bottom and bottom-to-top pruning appear directly relevant and may not be limited to the way we apply them with LL.

2 OVERVIEW
We start with some simplifying assumptions. Let k be the optimum of the given instance, which we guess via a binary search framework. If k ≤ poly(log log n), then obtaining a (1/k)-approximation is sufficient. This is easy to achieve: it is enough to find |S| vertex-disjoint paths that connect each source to a sink, which can be done by a standard max-flow algorithm. Throughout the paper we assume without loss of generality that k is at least poly(log log n) with sufficiently large constants. Similarly, we assume that n is larger than a sufficiently large constant. Whenever we divide k by some term, for example, k/log log n, we would normally have to write [k/log log n].

All the divisors considered can be assumed to be much smaller than k, hence any loss due to rounding is only a small constant factor and therefore insignificant. We assume for simplicity that the divisors are always integral and the divisions have no remainder.

We describe solutions using a set of paths instead of directly as arborescences. This abstraction will become useful in the linear programming relaxation, but also in other parts throughout the paper. With a given arborescence, we associate the set of all paths from a source to some vertex in it. Let p be a path from a source to some vertex v. Formally, p is a tuple of vertices (v0, v1,..., vn) where v0 is a source, v1 a vertex with an edge from v0, and more generally vn is a vertex having an edge from vn−1. We do not explicitly forbid circles, but the properties of a solution will imply that only simple paths can be used. Finally we say that p is a closed path if its last vertex is a sink and an open path otherwise.

We will denote by p ∪ v the path p, to which we add the extra vertex v at the end. We write |p| for the length of path p. A path q is said to be a descendant of p if it contains p as a prefix. In that case we call p an ancestor of q. We also say that p ∪ v is a child of p (who is then a parent of p ∪ v). Within a given set of paths P, we denote the sets of children, parent, ancestors, and descendants of a path p by C_P(p), P_p(p), A_p(p), and D_p respectively. By D_p(p, ℓ) we denote the descendants q ∈ D_p(p) with |q| = |p| + ℓ. Furthermore, we write D_p(p, ℓ) = ∪_{ℓ ≤ ℓ'} D_p(p, ℓ'). The set of paths that ends at a vertex v is denoted by P_v. If the set of paths P is clear from the context, we may choose to omit the subscript for convenience.

The conditions for a set of paths Q to form a degree-k solution are the following:
1. (s) ∈ Q for every source s,
2. |P_v| ≤ 1 for every v ∈ V, and
3. |D_p(p)| = k for every path p ∈ Q that is open.

We will also consider a relaxed version of (2), where we allow higher values than 1. Then we call maximum over all |H_ν| the congestion of the solution. The motivation for looking at solutions with (low) congestion is that we can remove any congestion by reducing k by the same factor.

Lemma 1. Let Q be a degree-k solution with congestion K. Then in polynomial time we can compute a degree-k/K solution without congestion.

Proof. Consider a bipartite multigraph (U ∪ U', F) on two copies U, U' of the vertices in V that appear in Q. The graph has an edge (u, v) ∈ F for every path p ∈ Q that ends in u and for which p ∪ v ∈ C(F). Then the degree of a vertex u' ∈ U' is exactly the congestion of this vertex. Similarly, the degree of a vertex u ∈ U is k times its congestion. We consider a fractional selection of edges x_ν = 1/K for each ν ∈ F. Here, each vertex u ∈ U has Σ_{ν ∈ δ(u)} x_ν ≥ k/K and each vertex u' ∈ U' has Σ_{ν ∈ δ(u')} x_ν ≤ 1, where δ(u) are the edges incident to u. As explained at the beginning of the section, we assume that K divides k and therefore K/k is integer. By integrality of the bipartite matching polytope there exists also an integral vector x' that satisfies these bounds. This corresponds to a degree-k/K solution without congestion. □

2.1 Bounded Depth Solution
The following proof follows exactly the arguments of a similar statement in [7]. We repeat it here for convenience.

Lemma 2. Let Q be a degree-k solution. Then there exists a degree-k/2 solution Q' ≤ Q where |p| ≤ log_k n for every p ∈ Q'.

Proof. We iteratively derive Q' from Q. For every d = 1, 2,..., n we consider the paths p ∈ Q with |p| = d. For d = 1 we add all these paths to Q'. Then given the paths of length d in Q' we select for each of them the subset of k/2 children in Q, which has the least number of descendants (assuming for simplicity that 2 divides k). For every d we will now bound the total number of descendants in Q of paths of this length, namely

n_d = Σ_{p ∈ Q': |p| = d} |D_Q(p)|.

Notice that the descendants are counted in set Q and not Q'. Clearly, we have n_1 ≤ n, since |Q| ≤ n. Then since we remove the children with the largest number of descendants, we get n_{i+1} ≤ n_i/2 for every i. Thus, n_{log_k n} = 0. □

2.2 Local Congestion and Layered Instances
A crucial concept in our algorithm are solutions with locally low congestion. It will later be shown that such a solution suffices to derive a solution with (globally) low congestion.
Definition 3. Let $Q$ be a solution and $t \in \mathbb{N}$. We say that $Q$ has an $\ell$-local congestion of $L$, if for every $p \in Q \cup \{\emptyset\}$ and $v \in V$ we have

$$|D(p, s(t)(v))| \leq L.$$ 

For sake of clarity, let us note the special case of $p = \emptyset$ in the definition above. In this case, $D(p, \leq t)$ is simply the set of all paths in $P$ with length at most $t$ (potentially starting at different sources). Throughout the paper we use values of the order $t = O(\log \log n)$ and $L = \text{poly}(\log \log n)$. The usefulness of local congestion is captured by the following lemma, which we will prove in Section 3.

Lemma 4. Let $Q$ be a degree-$k$ solution with $t$-local congestion of $L$ and global congestion $K$, where $t \geq \log_2 K$ and $K \geq \log_2 n$. Then we can compute in polynomial time a degree-$k/(8t)$ solution $Q' \subseteq Q$, which has global congestion at most

$$O(t^2 L).$$

It remains to show how to compute a solution with low local congestion. The abstraction of local congestion and the lemma on bounded depth allows us to reduce at a low expense to instances in layered graphs.

Definition 5. A layered instance has layers $L_0 \cup L_1 \cup \ldots \cup L_k = V$ such that $L_0$ consists of all sources and edges go only from one layer $L_i$ to the next layer $L_{i+1}$.

Lemma 6. In polynomial time we can construct a layered instance with $h = \log_2 n$ such that if there exists a degree-$k$ solution for the original instance, then there exists a degree-$k/2$ solution for the layered instance. Further, any degree-$k'$ solution with $t$-local congestion $L$ and global congestion $K$ in the layered instance can in polynomial time be transformed to a degree-$k''$ solution for the original instance with $t$-local congestion $\ell L$ and global congestion $K \log_2 n$.

Proof. Let $L_0$ be the set of sources. Then for each $i = 1, 2, \ldots, \log n$ let $L_i$ be a copy of all vertices $V$. We introduce an edge from $u \in L_i$ to $v \in L_{i+1}$ if $(u, v)$ is an edge in the original instance. The new set of sinks is the union of all sink vertices in all copies.

Consider now a degree-$k$ solution $Q$ for the original instance. By Lemma 2 there exists a degree-$k/2$ solution $Q'$ where each $p \in Q'$ has $|p| \leq \log_2 n$. For each such $p = (v_1, v_2, \ldots, v_{\ell})$ we introduce a path $p' = (v'_1, v'_2, \ldots, v'_{\ell})$ in the layered instance where $v'_{i}$ is the copy of $v_i$ in $L_i$.

Now let $Q'$ be a degree-$k'$ solution with $t$-local congestion $L$ and global congestion $K$ in the layered instance. We transform $Q'$ to a solution $\hat{Q}$ for the original instance by replacing each path $p' = (v'_1, v'_2, \ldots, v'_{\ell})$ by a path $p = (v_1, v_2, \ldots, v_{\ell})$, where $v'_i$ is a copy of $v_i$. Since there are only $\log_2 n$ copies of each vertex, the global congestion increases by at most a factor of $\log_2 n$. For the local congestion consider a path $p \in \hat{Q}$. This path was derived from a path $p' \in Q'$. Notice that any vertex $v'_q \in D(p', \leq t)$ ends in some vertex in $L_{|p'|+1}, L_{|p'|+2}, \ldots, L_{|p'|+t}$. Thus, there are only $t$ copies of each vertex that $q$ can end in. Consequently, the $t$-local congestion can increase at most by a factor of $t$. \hfill \Box

Lemma 7. Let $K = 2^{11} \log_2 n$, $t = 10 \log \log n$, and $L = \Omega(\log t^2)$. Given a layered instance with a single source and optimum $k$, we can in quasi-polynomial time compute a degree-$k/(64t)$ solution with $t$-local congestion at most $L$ and global congestion at most $K$.

This lemma is proven in Section 4. The lemmas above would allow us already to obtain our main result for instances with a single source. To generalize to multiple sources we present a black-box reduction on layered instances. This is proved in Section 5.

Lemma 8. Suppose in quasi-polynomial time we can compute an $\alpha$-approximation for the max-min degree bounded arborescence problem on layered instances with a single source. Then there is also a quasi-polynomial time $256\alpha$-approximation for layered graphs and an arbitrary number of sources.

2.3 Connecting the Dots

Theorem 9. We can compute a poly($\log \log n$)-approximation for the max-min degree arborescence problem in quasi-polynomial time.

Proof. First we prove the theorem on layered instances with a single source. Let $k$ be the optimum of the given instance. Using Lemma 7 we can find a degree-$\Omega(k/t)$ solution with $t$-local congestion $L$ and global congestion $K$. Here $K = O(\log^3 n)$, $t = O(\log \log n)$ with $2^{\theta} \geq K$, and $L = O(t^2)$. Next, we apply Lemma 4 to turn this into a degree-$\Omega(k/t^2)$ solution with global congestion at most $O(t^2 L) = O(\hat{L})$. Using Lemma 1 we can convert this to a degree-$\Omega(k/3t^2)$ solution without congestion. We therefore have an $O(\ell_{11})$-approximation algorithm for a single source on layered graphs and Lemma 8 implies that we can extend this to an arbitrary number of sources.

We now turn our attention to instances that are not necessarily layered. Let again $k$ be the optimum. Using Lemma 6 we construct a layered instance that is guaranteed to contain a degree-$k/2$ solution. Thus, with our algorithm for layered instances we can obtain a degree-$\Omega(k/\ell_{11})$ solution for it. This solution has $t$-local congestion at most $1$ and global congestion at most $1$. Using Lemma 6 we can construct a degree-$\Omega(k/\ell_{12})$ solution for the original (non-layered) instance with $t$-local congestion at most $t$ and global congestion at most $\log n$. Using again Lemma 4 we obtain a degree-$\Omega(k/\ell_{12})$ solution with global congestion at most $O(\hat{L})$. Finally, applying Lemma 1 we obtain a degree-$\Omega(k/\ell_{20})$ solution without congestion. In particular, our approximation ratio is

$$O(t^{20}) = \text{poly}(\log \log n).$$ \hfill \Box

2.4 Bottom-to-Top Pruning

In this subsection we will describe a method of pruning that is used in the proofs of Lemmas 4 and 7.

Suppose we are given a degree-$k$ solution and we want to remove some of the paths, for example, because they cause high congestion. This may lead to some other paths having few children and consequently these need to be removed as well. In general, even a very small amount of removals can lead to the whole solution getting corrupted, that is, ultimately sources may need to be removed as well. Below, we present a condition that guarantees that certain parts of the solution remain intact.

Lemma 10. Let $Q$ be a degree-$k$ solution. Let $R \subseteq Q$ be a set of paths that is supposed to be removed. Let $t \geq 2$ such that for every $p \in Q \setminus R$ we have that at most $k^t/8t^2$ many descendants $q \in D(p, t)$ with $q \in R$. Then we can compute in polynomial time a solution $Q' \subseteq Q \setminus R$ such that...
(1) $Q'$ is a degree-$k/2t$ solution and
(2) we have $(s) \in Q'$ for every source $s$, such that for any distance $t' \leq t$ there are at most $k^{t'}/(8t)$ many $q \in D((s), t')$ with $q \in R$.

Proof. We assume that no path $p \in R$ has a descendant also in $R$. This is without loss of generality, since removing the former implies that the latter will be removed, and omitting the latter from $R$ still keeps the premise of the lemma valid. In particular, this assumption allows us to assert that also paths in $R$ satisfy the bound on the number of descendants in $R$.

We prune the solution from longest paths to shortest paths: We remove a path if it is in $R$ or if more than $(1 - 1/2t)k$ many of its children were removed. Then we prove a stronger variant of (2) inductively, namely, that any path of length $1, 1 + t, 1 + 2t, \ldots$ satisfies the implication (or an ancestor of it is removed). Let $p$ be a path $[|p| = 1 + t \cdot t]$ that satisfies the premise of (2), but is not necessarily a singleton. Further, assume that all paths of length $1 + (t + 1)\ell$ satisfy the implication of (2). Let $t' \leq t$. Each of the distance-$t'$ descendants of $p$ has at most $(8\ell)^{-2}k^t$ many distance-$t$ descendants in $R$. Consequently, $p$ has at most $k^{t'} \cdot (8\ell)^{-2}k^t$ distance $(t + t')$-descendants in $R$. Thus, at most $(8\ell)^{-1}k^t$ many of $p$'s distance-$t$ descendants have more than $(8\ell)^{-1}k^t$ distance-$t'$ descendants belonging to $R$. Summing over all values of $t'$, we have that at most $1/8 \cdot k^t$ many distance-$t$ descendants of $p$ do not satisfy the premise of (2). Next, let us show in a second induction that for every $t' \leq t$, of the distance-$t'$ descendants of $p$ at most $\frac{1}{8} \left(1 + \frac{2}{t}\right)^{t-t'+1} k^t$ many are removed. For the base case we sum the distance-$t$ descendants that do not satisfy the premise of (2) and the descendants that are themselves in $R$, which together are at most $\frac{1}{8} k^t + \frac{1}{(8\ell)^2} k^t \leq \frac{1}{8} \left(1 + \frac{2}{t}\right)^{t-t'+1} k^t$.

Now assume that we removed at most $1/8 \cdot (1 + 2/t)\ell - \ell^t k^{t+1}$ paths from the distance-$(t' + 1)$ descendants. For each distance-$t'$ descendants that we remove because of few remaining children, there are $(1 - 1/(2\ell))k$ many distance-$(t' + 1)$ descendants that we removed. The bound from this and the number of distance-$t'$ descendants in $R$ lets us bound the number of distance-$t'$ descendants that we remove by $\frac{1}{k} \left(1 - \frac{1}{2\ell}\right)^{-1} \cdot \frac{1}{8} \left(1 + \frac{2}{t}\right)^{t-t'+1} k^{t+1} \leq \frac{1}{8} \left(1 + \frac{2}{t}\right)^{t-t'+1} k^t$.

It follows with $t' = 1$ that we remove at most $1/8 \cdot (1 + 2/t)\ell \leq e^2/8 \cdot k < (1 - 1/(2\ell)) k$ children of $p$. Hence, $p$ is not removed itself by the procedure. $\square$

2.5 Probabilistic Lemmas

We recall here two probabilistic results that we use extensively.

**Lemma 11 (Chernoff’s bound, see [14]).** Let $X_1, \ldots, X_n$ be independent random variables that take value in $[0, a]$ for some fixed $a$. Let $S_n = \sum_{i=1}^n X_i$. Then we have, for any $\delta \geq 0$,

$$
\Pr \left[ S_n \geq (1 + \delta) \mathbb{E}[S_n] \right] \leq \exp \left( \frac{-2\delta^2 \mathbb{E}[S_n]}{(2 + \delta)a} \right).
$$

**Lemma 12 (Constructive Lovász Local Lemma, see [26]).** Let $X$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. For any $A \in \mathcal{A}$, let $\Gamma_A (\mathcal{A})$ be the set of events $B \in \mathcal{A}$ such that $A$ and $B$ depend on at least one common variable. If there exists an assignment of reals $x: \mathcal{A} \mapsto (0, 1)$ such that for all $A \in \mathcal{A}$,

$$
\Pr [A] \leq x(A) \cdot \prod_{B \in \Gamma_A (\mathcal{A})} (1 - x(B))
$$

then there exists an assignment of values to the variables $X$ not triggering any of the events in $\mathcal{A}$. Moreover, there exists a randomized algorithm that finds such an assignment in expected time $|X| \sum_{A \in \mathcal{A}} x(A) / 1 - x(A)$.

3 LOCAL TO GLOBAL CONGESTION

This section is to prove Lemma 4. Let $Q$ be a degree-$k$ solution with $t$-local congestion $L$ and global congestion $K$. We partition $Q$ by length: let $Q_i$ be the set of paths $p \in Q$ with $|p| = i$. Further, we split the paths into $t$ groups $G_1, G_2, \ldots, G_t$, where $G_i = Q_i \cup Q_{i+1} \cup Q_{i+2} \cup \cdots$. For some $p \in G_i$, we write $G(p) = G_j$. Roughly speaking, we proceed as follows. We sample from each $G_i$ half of the paths and throw away all others (including their descendants). Then we move to $G_{i+1}$ and do the same. We continue until $G_t$ and then repeat the same a second time, stopping afterwards. The sampling is done in a way that guarantees that each path retains a quarter of its children at the end. We prove with Lovász Local Lemma that in each step we can reduce the congestion significantly. Since it seems unclear how to argue directly about the worst case congestion, we will argue about the congestion aggregated over many paths, which we will formalize next.

Consider a path $p \in Q$ and its close descendants in $D(p, \leq t)$. Recall, that $D(p, \leq t)$ contains all descendants of length at most $|p| + t$. Intuitively, if many of the direct children of $p$ have high congestion (more precisely, the vertex that they end in), this is bad for $p$ as well: if they have high congestion, we may not be able to keep many of these children for $p$, which means we might not be able to include $p$ itself in the solution. Let $\concg_\mathcal{P}(p)$ be the congestion of the last vertex in $p$, but restricted to paths in the group $G(p)$. In other words,

$$
\concg_\mathcal{P}(p) = \{|q \in G(p) \mid q \text{ ends in the same vertex as } p|\}.
$$

The restriction to other paths in $G(p)$ is only for technical reasons and almost at no cost: if we can achieve that every vertex is used by only few paths in each $G_j$ (i.e., $\concg_\mathcal{P}(p)$ is small for all $p \in Q$), the overall congestion can only be worse by a factor $t$. An important quantity in the following will be the total congestion of descendants $D(p, t')$ of $p$ at some distance $t' < t$, that is,

$$
\concg_\mathcal{P}(D(p, t')) = \sum_{q \in D(p, t')} \concg_\mathcal{P}(q),
$$

(1)
Since during the procedure the number of children may differ between groups $G_j$, we will use $k(G_j)$ to describe the current number of children for every open path in $G_j$. Our intermediate goal will be to bound the totals (1) for some $p \in G_j$ in terms of $k(p, t') = \prod_{j \in J}^t k(G_j)$ (an upper bound on $|D(p, t')|$).

Notice that initially (1) can be at most $K \cdot k(p, t')$. When sampling down the paths in $G_j$, we want to show that this reduces (1) significantly for paths $p \in G_j$ and all $t' < t$. This is captured in the following lemma.

**Lemma 13.** Assume we are given a solution that has an $\ell$-local congestion of at most $L$ and global congestion of at most $K$, where $\ell \geq \log K$ and $K \geq \log n$. From the paths in $G_j$ form pairs where each pair shares the same parent. Then select i.i.d. one path from each pair and remove it and all its descendants. Let $c_{G_j}$ and $c'_{G_j}$ be the congestion count before and after the removal and similarly $k$ and $k'$ the children count. Then we have with positive probability for every remaining $p \in G_j$ and $t' < t$ that

$$\frac{c_{G_j}(D(p, t'))}{k'(p, t')} \leq c\ell^2 L \frac{1}{2} \left(1 + \frac{1}{\ell}ight) \frac{c_{G_j}(D(p, t'))}{k(p, t')} ,$$

where $c$ is a fixed constant. Furthermore, we can obtain such a sampling in expected polynomial time.

**Proof.** We can rewrite

$$c_{G_j}(D(p, t')) = \sum_{q \in G_j(p)} c_{G_j}(D(p, t'), D(q, t')) ,$$

where $c_{G_j}(D(p, t'), D(q, t'))$ is the number of pairs $p' \in D(p, t'), q' \in D(q, t')$ that end in the same vertex. We remove every term in the sum with probability $1/2$, so in expectation the sum will reduce by $1/2$. Furthermore, each term $c_{G_j}(D(p, t'), D(q, t'))$ is bounded by $L \cdot |D(p, t')| \leq L \cdot k(p, t')$, because we have low local congestion. This will give us good concentration. Notice also that $k(p, t') = k'(p, t')$.

We now group the terms in the sum by their size. For $p \in G_j$ let $G_j(p, t', t)$ be the set of $q \in G_j(p) = G_j$ with $c_{G_j}(D(p, t'), D(q, t')) \in \{L \cdot k(p, t') \cdot 2^{-(t+1)}, L \cdot k(p, t') \cdot 2^{-t}\}$. Let $Q'$ be the set of paths remaining after sampling down (without taking into account those that are removed recursively). Let $c$ be a large constant to be specified later. Depending on whether $t$ is small or large, we define bad events $B(p, t', t)$ for each $p \in G_j$ as

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t'))$$

if $t \leq 2t$ and

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t'))$$

if $t > 2t$. Since from the experiment the total congestion cannot increase, we have a probability of 0 for all bad events where

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t')) \leq \left(\frac{12c\ell^2 L \cdot k(p, t')}{K}ight)^{t} \text{ if } t \leq 2t,$$

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t')) \leq \left(\frac{12c\ell^2 L \cdot k(p, t')}{K}ight)^{t} \text{ if } t \leq 2t,$$

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t')) \leq \left(\frac{12c\ell^2 L \cdot k(p, t')}{K}ight)^{t} \text{ if } t \leq 2t,$$

$$\sum_{q \in G_j(p, t', t) \cap Q'} c_{G_j}(D(p, t'), D(q, t')) \leq \left(\frac{12c\ell^2 L \cdot k(p, t')}{K}ight)^{t} \text{ if } t \leq 2t,$$

For the remaining bad events, we now derive an upper bound on the probabilities. This holds trivially also for the zero probability events. From Chernoff’s bound we get

$$P[B(p, t', t)] \leq \left(\sum_{q \in G_j(p, t', t)} c_{G_j}(D(p, t'), D(q, t')) \right) \leq \left(\frac{12c\ell^2 L \cdot k(p, t')}{K}ight)^{t} \text{ if } t \leq 2t.$$
Hence, by LLL we have with positive probability none of the bad events occur. If none of them occur, then by summing up bounds fixed $p$ and $t'$ we get
\[
\text{cong}(D(p, t')) \leq 2t \cdot 24\epsilon t^2 L \cdot k(p, t') + \log n \cdot 24\epsilon t^3 L \cdot k(p, t')/K + \frac{1}{2} \left(1 + \frac{1}{\ell} \right) \text{cong}(D(p, t'))
\]
\[
\leq 100\epsilon t^4 L \cdot k(p, t') + \frac{1}{2} \left(1 + \frac{1}{\ell} \right) \text{cong}(D(p, t'))
\]
\[
\text{Lemma 14. Consider a successful run of the random experiment in Lemma 13 where we sample down the paths in $G_j$ and satisfy the stated inequalities. For each path $p$ (potentially not in $G_j$) and every $t' \in \{1, 2, \ldots, \ell\}$ we have}
\[
\frac{\text{cong}_G(p, t')}{k'(p, t')} \leq c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{\text{cong}_G(p, t')}{k'(p, t')}
\]
Here $c$ is the constant from Lemma 13.

Lemma 13 only shows that the average congestion reduces for descendants of paths in the group $G_j$, where we sample down. Conversely, Lemma 14 says that for all other groups it does not increase significantly.

Proof. Let $G_{j'} = G(p)$. We can assume without loss of generality that $j' < j \leq j' + t'$ (modulo $t$), since the congestion can only decrease and $k(p, t')$ in the other case would not change. Let $t'' = j - j'$ (modulo $t$). Further, let $D'(p, t')$ be the distance-$t'$-descendants of $p$ after sampling down $G_j$ and $D(p, t')$ before it. Then $D'(p, t'')$ contains half of the elements $D(p, t'')$. Thus,
\[
\text{cong}_G(p, t') \leq k'(p, t') \sum_{q \in D'(p, t'')} \text{cong}_G(q, t' - t'') \leq \frac{1}{k(p, t')} \sum_{q \in D(p, t'')} \text{cong}_G(q, t' - t'') \leq \frac{2}{k(p, t')} \sum_{q \in D(p, t'')} \left[\epsilon t^4 L + \frac{1}{2} \left(1 + \frac{1}{\ell} \right) \text{cong}_G(q, t' - t'')\right] \leq c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \text{cong}_G(p, t')
\]
\[
\text{Lemma 15. Given a degree-$k$ solution $Q$ with $t$-local congestion $L$ and global congestion $K$, we can compute a degree-$k/4$ solution $Q' \subseteq Q$ with}
\[
\text{cong}_{G_k}(p, t) \leq 3c\epsilon t^4 L \left(\frac{k}{4}\right)^t + \left(1 + \frac{1}{\ell} \right) \frac{K}{2^t}
\]
for all $p \in Q$. Here $c$ is the constant from Lemma 13.

Proof. We will perform the sampling from Lemma 13 for $G_1, G_{t-1}, \ldots, G_1$ and then again the same a second time. The reason is that we want that for every group $G_j$ that $G_j, G_{j-1}, \ldots, G_{j-\ell}$ (index modulo $\ell$) are down-sampled at least once in this order.

Let $G_j = G(p)$ and consider the first time that we sample down $G_{i+\ell-1}$ (index taken modulo $\ell$). Let $D(p, t')$ be the distance-$t'$-descendants of $p$ before this sampling. Then for each $q \in D(p, t - 1)$ we have
\[
\frac{\text{cong}_G(D(q, 1))}{k(q, 1)} \leq K
\]
This is due to the fact that sampling down cannot increase the congestion on any vertex and initially all vertices have congestion at most $K$. After sampling according to Lemma 13 we have
\[
\frac{\text{cong}_G(D(q, 1))}{k(q, 1)} \leq c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{K}{2}
\]
In the next step we are sampling down $G_{j+t-2}$. We have for each $q \in D(p, t - 2)$ that
\[
\frac{\text{cong}_G(D(q, 2))}{k(q, 2)} \leq \frac{1}{k(q, 1)} \sum_{q' \in C(q)} \frac{\text{cong}_G(q', 1)}{k(q', 1)} \leq c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{K}{2}
\]
Thus, after sampling
\[
\frac{\text{cong}_G(D(q, 2))}{k(q, 2)} \leq c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right)^2 \frac{K}{4}
\]
Continuing this argument, after we sample down $G_j$ we have
\[
\frac{\text{cong}_G(p, t)}{k(p, t)} \leq 3c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{K}{2^t}
\]
After $G_j$ there may be at most $t$ more steps of sampling down, after which we finally have
\[
\frac{\text{cong}_G(p, t)}{k(p, t)} \leq 3c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{2t}{2^t} \frac{K}{2^t} \leq 3c\epsilon t^4 L + \left(1 + \frac{1}{\ell} \right) \frac{2}{2^t} \frac{K}{2^t}
\]
We will now conclude the proof of Lemma 4. Using the previous lemma, we obtain a degree-$k/4$ solution $Q'$. Let $A := 3c\epsilon t^4 L + \left(1 + 1/\ell^2\right) K/2^t$ be the upper bound on average distance-$t$ congestion. Assuming without loss of generality that $c$ is sufficiently large, we have that $A \leq 3c\epsilon t^4 L$. Let $R$ be the set of all paths $p$ with $\text{cong}_G(p) > 16A\epsilon^2 t$. We remove these paths using Lemma 10. We use the property that each path has at most a $(8\ell)^{-2} (k/4)^t$ many of its ancestors at distance $\ell$ in $R$. This follows directly from the bounded average congestion. The lemma implies that we can remove those high congestion paths and still keep a solution where each remaining path has $k/(4 \cdot 2\ell) = k/8\ell$ children. Since we start with a $\ell$-local congestion of at most $L \leq 16A\epsilon^2 t$ and this cannot be increased by only removing paths, we have that none of the paths of length at most $\ell$ are removed and thus all sources satisfy the premise of (2) of Lemma 10 and remain in the solution. Indeed, the value of $\text{cong}_G(p)$ is now bounded by $16A\epsilon^2 t$ for all remaining paths. We recall that the actual congestion is at most a factor $\ell$ higher than $\max_p \text{cong}_G(p)$, that is,
\[
16A\epsilon^2 t \leq 64c\epsilon^2 t^3 L.
\]
4 COMPUTING A SOLUTION WITH LOCALLY LOW CONGESTION

The goal of this section is to prove Lemma 7. We recall that we are in a layered graph with vertices partitioned into layers $L_0, L_1, \ldots, L_h$ where $h = \log n$ and a single source $s$. The source $s$ belong to layer $L_0$ and edges can only go from vertices in some layer $L_i$ to the next layer $L_{i+1}$.

In the following we will extensively argue about paths that start in the source. For the remainder of the section every path $p$ that we consider is implicitly assumed to start at the source and then traverse (a prefix of) the layers one by one. Slightly abusing notation, we sometimes use $L_i$ also to denote the set of paths ending in a vertex of $L_i$, that is,

$$\bigcup_{v \in L_i} I(v),$$

and $T$ to describe the set of closed paths (recall that those are the paths that end at a sink). Let $P$ refer to the set of all possible paths and notice that by virtue of the layers we have that $|P| \leq n^h \leq n^{\log n + 1}$. We will now describe a linear programming relaxation, which goes back to Bateni et al. [7]. The intuition behind the linear program is to select paths similarly to the way we describe solutions, see Section 2. We have a variable $x(p)$ for each path $p$ that in an integral solution takes value $1$ if the path $p$ is contained in an arborescence and $0$ otherwise.

$$\sum_{q \in C(p)} x(q) = k \cdot x(p) \quad \forall p \in P \setminus T \quad (2)$$

$$\sum_{q \in I(v) \cap D(p)} x(q) \leq x(p) \quad \forall p \in P, v \in V \quad (3)$$

$$x(s) = 1 \quad (4)$$

$$x \geq 0 \quad (5)$$

Here we assume that $k$ is the highest value for which the linear program is feasible, obtained using a standard binary search framework. Moreover, we assume that $k \geq 2^{10(\log \log n)^3}$ in the rest of the section. This is at little cost, since a $1/k$-approximation is easy to obtain (see the proof of Theorem 9) and is already sufficient for our purposes. The first two types of constraints describe that each open path has many children and each vertex has low congestion (in fact, no congestion). Constraint (3) comes from a lift-and-project idea. For integral solutions it would be implied by the other constraints, but without it, there could be situations with continuous variables, where for example we take a path $p$ with only value $1/k$ and then a single child of $q$ with value $1$. Such situations easily lead to large integrality gaps, which we can avoid by this constraint.

Since the graph has $h + 1$ many layers, this linear program has $\mathcal{O}(h)$ variables and constraints and therefore can be solved in time $\mathcal{O}(h)$. We refer to this relaxation as the path LP. In order to prove Lemma 7, we will design a rounding scheme. Before getting to the main part of the proof, we will first preprocess the fractional solution to sparsify its support.

4.1 Preprocessing the LP Solution

Our first step is to sparsify the path LP solution $x$ to get another sparser solution (i.e., with a limited number of non-zero entries). For ease of notation, we might need to take several times a copy of the same path $p \in P$. We emphasize here that two copies of the same path are different objects. To make this clear, we will now have a multiset $P'$ of paths but we will slightly change the parent/child relationship between paths. Precisely, for any path $q' \in P'$ is assigned as child to a unique copy $p' \in P'$ such that $q$ was a child of $p$ in the set $P$. With this slight twist, all the ancestors/descendants relationships extend to multisets in the natural way. For instance, we will denote by $D_{p'}(p)$ the set of descendants of $p$ in the multiset $P'$. Again, the set of closed path in $P'$ will be denoted by $T'$ and $s$ refers to the source. We assume that there is a unique copy of the trivial path $(s) \in P'$.

In this step we will select paths such that each open path has $k \log^2 n$ children instead of the $k$ children one would expect. However, we use a function $y(p)$ that assigns a weight to each path and this weight decreases from layer to layer, modelling that the children are actually picked fractionally each with a $1/4 \log^2 n$ fraction of the weight of the parent. Thus, taking the weights into account we are actually picking only $k/4$ children for each path.

Formally, the preprocessing of the LP will allow us to obtain a multiset of path $P'$ such that

$$\sum_{q \in C_p(p)} y(q) = \frac{k}{4} \cdot y(p) \quad \forall p \in P \setminus T' \quad (6)$$

$$\sum_{q \in I_v(p) \cap D_{p'}(p)} y(q) \leq 2y(p) \quad \forall p \in P', v \in V \quad (7)$$

$$y(p) = \frac{1}{(4 \log^2 n)^{\ell}} \quad \forall i \leq h, \forall p \in L_i \quad (8)$$

We obtain such a solution $P'$ in a similar way as the randomized rounding in [7, 9], which achieves polylogarithmic congestion. The fact that we select more paths, but only fractionally gives us better concentration bounds, which allows us to lose only constant congestion here. For the proof we refer to the full version of the paper.

4.2 The Main Rounding

We start this part with the sparse multiset of paths $P'$ with the properties as above. The discount value $y$ can be thought of fractionality in the sense that each $p \in P'$ is taken to an extend of $y(p)$. We will proceed to round this fractional solution to an integral solution $Q$ that is locally nearly good, a concept we will define formally below. Intuitively, this specifies that the number of paths that have locally high congestion can be removed without losing much with Lemma 10. We fix $\ell = 10 \log \log n$ for the rest of this section. Let $\text{cong}(v \mid Q)$ be the global congestion of vertex $v$ induced by $Q$, that is, the number of paths in $Q$ ending in $v$. For paths $p \in Q \cup \{\emptyset\}$ and $q \in D_{Q}(p)$ we denote by $\text{cong}_{p}(q \mid Q)$ the local congestion induced on the endpoint of $q$ by descendants of $p$. We consider all paths descendants of $\emptyset$. A locally nearly good solution is a multiset of paths $Q \subseteq P'$ (where again every path has a unique parent among the relevant copies of the same path) that has the following properties:

1. one copy of the trivial path $(s)$ belongs to $Q$;
2. every open path has $k/32$ children;
3. no vertex has global congestion more than $2^{10} \log^3(n)$;
(4) for every \( p \in Q \cup \{0\} \) and \( t' \leq t \) we have
\[
|\{ q \in D_Q(p, t') \mid \text{cong}_p(q \mid Q) > 2^{10t^2} \}| \leq \frac{k^{t'}}{32}.
\]
From a locally nearly good solution we will then derive a solution of low local congestion by removing all paths with high local congestion using bottom to top pruning (Lemma 10). Condition 4 is tailored to ensure that the number of such high local congestion paths is small enough so that the lemma succeeds.

To obtain such a nearly good solution, we will proceed layer by layer, where the top layers are already rounded integrally and the bottom layers are still fractional (as in the preprocessing). However, unlike the case of the preprocessing, we cannot argue with high probability and union bounds, since some properties we want only deviate from expectation by \( \log(n) \) factors. To obtain such locally nearly good solution, we will use Lovász Local Lemma (LLL) in every iteration, where one iteration rounds one more layer. In the following, we describe the rounding procedure, the bad events and an analysis of their dependencies and finally we apply LLL.

The Randomized Rounding Procedure. We proceed layer by layer to round the solution \( P' \) (with discount \( y \)) to an integral solution \( Q \). We start by adding a single copy the trivial path \( s \) to the partial solution \( Q^0 \). Assume we rounded until layer \( i \), that is, we selected the final multiset of paths \( Q^{(i)} \) to be used from all paths in \( L_{\leq i} \).

To round until layer \( i + 1 \) we proceed as follows. Every open path \( p \in Q^{(i)} \cap L_i \) selects exactly \( k/16 \) children where the \( i \)-th child equals \( q \in C_p(p) \) with probability equal to
\[
\frac{1}{k \log^2 n} = \frac{y(q)}{\sum_{q' \in C_p(p)} y(q')}.
\]
The selection of each child is independent of the choices made for other children. We then let \( Q^{(i+1)} \) be the union of \( Q^{(i)} \) and all newly selected paths. The reason that each path selects \( k/16 \) children instead of the \( k/32 \) many that were mentioned before is that we will later lose half of the children. We will repeat this procedure until reaching the last layer \( h \) and we return \( Q = Q^{(h)} \).

Definitions Related to Expected Congestion. In order for this iterated rounding to succeed we need to avoid that vertices get high congestion (in the local and the global sense). It is not enough to keep track only of the congestion of vertices in the next layer that we are about to round, but we also need to maintain that the expected congestion (over the remaining iterations) remains low on all vertices in later layers. Hence, we define quantities that help us keep track of them. To avoid confusion, we remark that the quantities we will define do not exactly correspond to the expected congestion of the vertices: notice that we sample less children for each path than \( P' \) has. Intuitively, \( P' \) has \( k/4 \) children per open path (see Equation (6)), but we only sample \( k/16 \) many. The quantities we define below would be the same if we would sample \( k/4 \) children instead. Roughly speaking, this gives us an advantage of the form that the expectation always decreases by a factor \( 4 \) when we round one iteration. Apart from the intuition on the expectation, the definitions also incorporate a form of conditional congestion, which means that we consider the (expected) congestion based on random choices made so far. This is similar to notions that are needed in the preprocessing step.

First, we define the fractional congestion induced by some path \( p \in P' \) on a vertex \( u \in V \) as follows.
\[
\text{cong}(p, u) := \sum_{q \in D_p(p') \cap I(p, u)} y(q).
\]
Using this definition, we will define the conditional fractional congestion at step \( i \) induced by \( p \) on a descendant \( q \) as follows (writing \( v_q \) for the endpoint of \( q \)).
\[
\text{cong}_p(q \mid Q^{(i)}) := \frac{\sum_{q' \in D_p(q) \cap L_i} y(q') \cdot \text{cong}(q', v_q)}{y(q')}. \text{ if } q \in L_{\leq i}
\]
\[
\text{ otherwise.}
\]
We will use this definition only for \( p \in Q^{(i)} \). If \( v_q \) belongs to one of the first \( i \) layers (i.e., \( v_q \) is in the integral part corresponding to the partial solution), this is simply the number of descendants of \( p \in Q^{(i)} \) in our partial solution that end at \( v_q \). Otherwise, this is the total fractional congestion induced by all the paths that we actually selected in our partial solution \( q \in Q^{(i)} \cap L_i \) and that are descendants of \( p \). The name conditional comes from the term \( 1/y(q') \) which is simply the fact that we condition on having already selected the path \( q' \) once integrally. The global congestion at step \( i \) on a vertex \( v \in V \) is defined similarly with
\[
\text{cong}(v \mid Q^{(i)}) := \frac{\sum_{q \in Q^{(i)} \cap L_i} y(q) \cdot \text{cong}(q, v)}{\sum_{q \in Q^{(i)} \cap L_i} y(q)} \text{ if } v \in L_{\leq i}
\]
\[
\text{ otherwise.}
\]

The First Type of Bad Event: Global Congestion. The naive way to bound the global congestion would be to simply to bound the global congestion on each vertex and make a bad event from exceeding this. However, to manage dependencies between bad events and get better concentration bounds, we partition the set of paths according to how much fractional load their ancestors are expected to put on \( v \) and then bound the congestion incurred by each group on \( v \). More precisely, assume we rounded until layer \( i \) and are now trying to round until layer \( L_{i+1} \). Consider any vertex \( v \in L_{i+1} \). The decisions made in this step of the rounding concern which paths in \( L_{i+1} \) will be selected. Fix an integer \( t \geq 0 \) and let \( P_{i+1}^{(t)} \) to be the set of all paths \( p \in L_{i+1} \) such that
\[
\frac{\text{cong}(p, u)}{y(p)} \in \left(2^{-t}, 2^{-(t-1)}\right).
\]
We define the bad event \( B_{i+1}(v, t) \) as the event that
\[
\sum_{p \in Q^{(i+1)} \cap P_{i+1}^{(t)}} \text{cong}(p, u) y(p) > 2E \left[ \sum_{p \in Q^{(i+1)} \cap P_{i+1}^{(t)}} \text{cong}(p, u) y(p) \right] + 2^{10} \log n.
\]
Since \( P_{i+1}^{(t)} \) partitions the paths in \( L_{i+1} \), the bad events for all \( t \) together will bound the increase in the congestion on \( v \): this is because \( \text{cong}(v \mid Q^{(i+1)}) = \sum_{p \in Q^{(i+1)} \cap P_{i+1}^{(t)}} \text{cong}(p, v) y(p) \). We argue this formally in Lemma 16 below.

At this point, it is worthwhile to mention that \( t \) can only take values in \( \{0, 1, \ldots, \log^2 n\} \). Indeed, by Constraint (7) we have that
\[
\frac{\text{cong}(p, v)}{y(p)} \leq 2
\]
is ensured for all \( p \in P' \). Moreover, we notice that \( y \) has a low granularity (i.e. \( y \) satisfies Equation (8)) which implies that either
cong\((p, v) = 0\) or
\[
\frac{\text{cong}(p, v)}{y(p)} \geq \frac{1}{(4 \log^2 n)^{h-1}} \geq \frac{1}{(4 \log^2 n)^h}.
\]
Hence, assuming here that \(n\) is sufficiently large, bad events \(B_1(v, t)\) will be instantiated only for \(t = 0, 1, \ldots, \log^2 n\).

**Lemma 16.** Assuming no bad event \(B_2(v, t)\) has occurred in any iteration up to \(i\), we have for every \(v\) that
\[
\text{cong}(v | Q^{(i)}) \leq 2^{11} \log^3 n.
\]

We refer to the full version of the paper for a proof of the lemma.

The Second Type of Bad Event: Local Congestion. Recall that we want to bound the congestion induced by close descendants of some open path \(p\). Let \(t' \leq t\) and \(p \in Q^{(i)} \cap L_{t'}\), where \(i - t' \leq i\). We define
\[
N^{(i+1)}(p, t') = \{q \in D_{p'}(p, t') | \text{cong}_p(q | Q^{(i)}) \leq 2^{10} t^2 \text{ and } \text{cong}_p(q | Q^{(i)}) > 2^{10} t^2\}.
\]
These are the number of vertices with newly high local congestion (counting only the congestion induced by descendants of \(p\)). Moreover, note that by Constraint (6) and (8), we have that \(|D_{p'}(p, t')| \leq (k \log^2 n)^{t'}\).

Then, for any \(p \in L_{t'} \cap Q^{(i)}\) where \(i - t - 1 \leq t' \leq i - 1\), and \(i \geq 0\), we define the set of marked children of \(p\) at step \(i + 1\) as the set \(M^{(i+1)}(p)\) of all \(q \in C_{Q^{(i)}}(p)\) such that
\[
N^{(i+1)}(q, t') > \frac{1}{\log^{10} n} \left(\frac{k}{32}\right)^{t'}
\]
for at least one \(t' \leq t\). Notice that the children of \(p\) (in \(Q^{(i)}\)) are already determined because \(p\) is in \(L_{t-1}\). We are now ready to state the second type of bad event. We define the bad event \(B_2(p)\) for any \(p \in L_{t'}\) where \(i - t - 1 \leq t' \leq i - 1\), as the event that
\[
|M^{(i+1)}(p)| \geq \frac{1}{\log^{10} n} \cdot \frac{k}{16},
\]
which means that a lot of children selected by \(p\) become marked at step \(i + 1\). As we will ensure that this bad event never happens, we can guarantee that a large fraction of children \(q\) of each path \(p\) always satisfies that \(N^{(i+1)}(q, t')\) is low for all \(t' \leq t\). As we explain later, we will simply remove all other children and this way indirectly prevent the existence of too many locally congested descendants for all the remaining paths. The advantage of this indirect method is that we can apply good concentration bounds on the children and thereby amplify the probability (indeed, note that the event that a child \(q\) of \(p\) is marked is independent of the same events for other children of \(p\)).

The Third Type of Bad Event: Keeping the Source Path (s). Because our algorithm will delete in the end all the paths that have been marked in some round, we need to ensure that the source path (s) is never marked. Hence for the first \(t\) steps of rounding, we define a single bad event \(B_3(s)\) that is that the source path (s) is marked during this step.

**The Probabilities of the Bad Events.** In this part, we derive upper bounds for the probability of each bad event. Some bounds are sub-optimal to simplify later formulas.

**Lemma 17.** For any \(v \in V\) and integer \(t\) we have that
\[
\mathbb{P}[B_1(v, t)] \leq n^{-10 \cdot 2^{t-1}}.
\]

**Proof.** We need to show that the event that
\[
\sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{y(p)}{\text{cong}_p(q | Q^{(i)})} > 2 \mathbb{E} \left[ \sum_{p \in Q^{(i+1)} \cap P_{v,t}^{(i+1)}} \frac{y(p)}{\text{cong}_p(q | Q^{(i)})} \right] + 2^{10} \log n
\]
induces with probability at most \(n^{-10 \cdot 2^{t-1}}\). To see this, notice that \(\text{cong}(p, v)/y(p)\) is upper bounded by \(2^{-(t-1)}\) by definition of \(P_{v,t}^{(i+1)}\) and the sum can be rewritten as a sum over the paths chosen in iteration \(i + 1\), where a path \(p\) contributes \(\text{cong}(p, v)/y(p)\) to the sum. These paths are chosen independently and thus by a Chernoff bound we obtain that the probability is at most
\[
\exp \left[ \frac{-2^{10} \log n}{2 \cdot 2^{-(t-1)}} \right] \leq n^{-10 \cdot 2^{t-1}}.
\]

**Lemma 18.** Let \(t' \leq t\) and \(p \in Q^{(i)} \cap L_{t'}\), where \(i - t' \leq i\). Then
\[
\mathbb{P}[N^{(i+1)}(p, t') > (1/\log^{10} n) \cdot (k/32)^{t'}] \leq (\log n)^{-40t}.
\]
Moreover,
\[
\mathbb{P}[B_3(s)] \leq (\log n)^{-1}.
\]

**Proof.** Recall that we are bounding the number of descendants of \(p\) that had low local congestion of at most \(2^{10} t^2\) before, but now get high local congestion. More concretely, consider some \(q \in D_{p'}(p, t') + \text{cong}_p(q | Q^{(i)}) \leq 2^{10} t^2\). Then
\[
\mathbb{E} \left[ \text{cong}_p(q | Q^{(i)}) \right] = \frac{1}{4} \cdot \text{cong}_p(q | Q^{(i)}) \leq 2^8 t^2.
\]
We do not go into detail for this calculation here but similar calculations have been derived already in Lemma 16. This is exactly the same type of argument.

Further, \(\text{cong}_p(q | Q^{(i+1)})\) can be decomposed into a sum of independent random variables bounded by \(2\). Indeed, recalling the definition of \(\text{cong}_p(q | Q^{(i+1)})\) we see that this quantity can be written as a sum over \(1/y(q') \cdot \text{cong}_q(q | Q^{(i+1)})\), where a paths \(q'\) are taken from a certain multiset of chosen paths. Constraint (7) of the path LP ensures that this is always at most \(2\). Those variables are independent because of the randomized rounding that selects children independently of each other. Hence, by a Chernoff bound we have
\[
\mathbb{P}[\text{cong}_p(q | Q^{(i+1)}) > 2^{10} t^2] \leq \exp(-2^7 t^2).
\]
Therefore, by linearity of expectation we obtain
\[
\mathbb{E}[N^{(i+1)}(p, t')] \leq \frac{(k \log^2 n)^{t'}}{\exp(2^7 t^2)} \leq \frac{1}{(\log n)^{2^7+10t}} \left(\frac{k}{32}\right)^{t'}.
\]
From Markov’s inequality it follows that
\[
\mathbb{P}[N^{(i+1)}(p, t') > (1/\log^{10} n) \cdot (k/32)^{t'}] \leq (\log n)^{-40t}.
\]
For the second claim, notice that \( B_3(s) \) is simply the event that the source path \( (s) \) is marked, which is equivalent to \( N^{(iv)}(s, t') > (1/\log n) \cdot (k/32)^t \) for at least one \( t' \leq t \). By a simple union bound, we get the desired result. \( \square \)

Finally, we upper-bound the probability of \( B_3(p) \).

**Lemma 19.** Let \( p \in L_t \) where \( i - t - 1 \leq t' \leq i - 1 \). Then we have that

\[
\Pr[ B_3(p) ] \leq \exp\left(-\sqrt{\frac{k}{2}}\right).
\]

**Proof.** To prove this, note that a child \( q \) of \( p \) (in the set \( Q^{(i)} \)) is marked independently of other children of \( p \). This is because \( q \) being marked depends only on the random choices made by descendants of \( q \), which are independent of descendants of other children of \( p \). Therefore using Lemma 18 and a standard union bound we obtain that each child of \( p \) is marked independently with probability at most \( \exp(-\sqrt{\frac{k}{2}}) \). Recall that \( k \geq 2^{10}(\log \log n)^8 \geq (32\sqrt{n})^2 \). We note that \( p \) has \( k/16 \) children in \( Q^{(i)} \) and therefore the probability that more than \( (1/\ell^3) \cdot (k/16) \) of them are marked is at most

\[
\exp\left(-\frac{k}{32\ell^3}\right) \leq \exp\left(-\sqrt{\frac{k}{2}}\right). \hspace{1cm} \square
\]

**The Dependencies of the Bad Events.** For any bad event \( B \) we define \( \Gamma_1(B) \) to be the set of bad events of the first type \( \mathcal{B}_4(v, t) \) that depend on the bad event \( B \). Similarly, let \( \Gamma_2(B) \) be the set of bad events \( \mathcal{B}_5(p) \) that depend on \( B \). We will now upper bound the cardinality of these sets. Note that there is a single bad event of third type \( \mathcal{B}_3(s) \) hence it is clear that \( \Gamma_3(B) \leq 1 \) for any bad event \( B \). For the rest, we remark that the focus here is on simplicity of the terms rather than optimizing the precise bounds.

**Lemma 20.** For any bad event \( B \), we have that

\[
|\Gamma_2(B)| \leq n^2.
\]

**Proof.** The statement holds trivially, since there are at most \( n \cdot (1 + \log^2 n) \leq n^2 \) bad events of the first type in total (\( n \) possibilities of the choice of vertex \( v \) and less than \( n \) possibilities for the choice of \( t \)). \( \square \)

Before proving the dependencies to events of type 2, we will prove an auxiliary lemma that concerns the events affected by the children picked by one particular path \( p \in Q^{(i)} \cap L_i \).

**Lemma 21.** Let \( p \in Q^{(i)} \cap L_i \). Then the choice of children picked by \( p \) affects in total at most \( \log \log n \) events of type 2.

**Proof.** In order to influence a bad event \( \mathcal{B}_2(q) \) for some \( q \), it must be that \( p \) is a descendant of \( q \) in the multiset \( Q^{(i)} \). Any path has at most \( h \leq \log n \) ancestors. \( \square \)

**Lemma 22.** Assuming that no bad event has happened until the current iteration of rounding, for any \( v \in V \) and integers \( t \geq 0 \), we have

\[
|\Gamma_2(\mathcal{B}_1(v, t))| \leq 2^t \cdot (\log n)^3. \hspace{1cm} (11)
\]

**Proof.** We first notice that the bad event \( \mathcal{B}_1(v, t) \) depends only on the random choices made by paths \( q \in Q^{(i)} \cap L_i \) that have at least one child \( q' \in C_p(q) \) such that

\[
\text{cong}(q', v) = \frac{y(q')}{y(q)} \geq 2^{-t}.
\]

Let us count how many such paths \( q \) can exist in \( Q^{(i)} \). For each such path we have that

\[
\text{cong}(q, v) = \sum_{q' \in C_p(q)} \frac{\text{cong}(q', v)}{y(q)} = \sum_{q' \in C_p(q)} \frac{|\text{cong}(q', v)|}{y(q)} \leq \frac{2^{-t}}{4 \log^2 n},
\]

where we used here the granularity of \( y \), see Constraint (8). Since we assume that no bad event happened so far, we can apply Lemma 16 to get

\[
\sum_{q \in Q^{(i)} \cap L_i} \frac{\text{cong}(q, v)}{y(q)} = \text{cong}(v \mid Q^{(i)}) \leq 2^{11} \log^3 n.
\]

Hence, there can be at most \( 2^t \cdot 2^{13} \log^5 n \) such paths \( q \). We conclude with Lemma 21 to bound the number of bad events of second type influenced by each \( q \) and obtain

\[
\left|\Gamma_2(\mathcal{B}_1(v, t))\right| \leq 2^t \cdot (2^{13} \log^5 n) \cdot (\log n) \leq 2^t \cdot (\log n)^8. \hspace{1cm} \square
\]

**Lemma 23.** For any \( p \in L_t \) where \( i - t - 1 \leq t' \leq i - 1 \) we have

\[
\left|\Gamma_2(\mathcal{B}_2(p))\right| \leq k^t \cdot (\log n). \hspace{1cm} (12)
\]

**Furthermore,**

\[
\left|\Gamma_2(\mathcal{B}_3(s))\right| \leq k^t \cdot (\log n). \hspace{1cm} (13)
\]

**Proof.** The bad event \( \mathcal{B}_2(p) \) (or \( \mathcal{B}_3(s) \)) is entirely determined by the random choices made by the descendants of \( p \) (or \( s \)) in \( Q^{(i)} \) that are located in layer \( L_i \). There are at most \( k/16 \) such descendants \( q \). By Lemma 21, each descendant can influence at most \( \log n \) other bad events of the second type. This concludes the proof. \( \square \)

**Instantiating LLL.** For the bad events we set

\[
x(\mathcal{B}_1(v, t)) = n^{-2}, \hspace{1cm} (14)
\]

\[
x(\mathcal{B}_2(p)) = \exp(-k^{1/3}) \hspace{1cm} (15)
\]

\[
x(\mathcal{B}_3(s)) = 1/2. \hspace{1cm} (16)
\]

Consider now a bad event \( \mathcal{B}_1(v, t) \) and recall that \( k \geq 2^{10}(\log \log n)^8 \). Then

\[
\Pr[ \mathcal{B}_1(v, t) ] \leq n^{-10 \cdot 2^{(t - 1)}}
\]

\[
\leq n^{-5 \cdot 2^t}
\]

\[
\leq n^{-2} (1 - n^{-2})^{n^2} \cdot \frac{1}{2} \left( 1 - \exp(-k^{1/3}) \right)^{2^t (\log n)^{10}}
\]

\[
\leq n^{-2} \prod_{2 \in \Gamma(\mathcal{B}_1(v, t))} (1 - x(\mathcal{B})).
\]

1873
Next, we get
\[
\mathbb{P}\left[\mathcal{B}_2(p)\right] \leq \exp\left(-\sqrt{k}\right)
\leq \exp\left(-k^{1/3}\right) \left(1 - \frac{n}{2}\right)^2 \cdot \frac{1}{2} \left(1 - \exp(-k^{1/3})\right)^{k^3/(\log n)}
\leq x(\mathcal{B}_2(p)) \cdot \prod_{B \in \mathcal{B}_1(\mathcal{B}_2(p))} (1 - x(B)).
\]

Similarly, for \(\mathcal{B}_3(s)\) we have
\[
\mathbb{P}\left[\mathcal{B}_3(s)\right] \leq (\log n)^{-1}
\leq \frac{1}{2} \left(1 - \frac{n}{2}\right)^2 \cdot \left(1 - \exp(-k^{1/3})\right)^{k^3/(\log n)}
\leq x(\mathcal{B}_3(s)) \cdot \prod_{B \in \mathcal{B}_1(\mathcal{B}_2(s))} (1 - x(B)).
\]

Hence, with positive probability none of the bad events occur and we can compute such a solution in expected quasi-polynomial time (polynomial in \(p^*)\).

**Finishing Notes.** We proved that we can round in (expected) quasi-polynomial time from layer 0 to layer \(h\) without any bad event ever happening. Next, observe that we get that any sampled path \(p \in Q \cap L_1\) has at most
\[
\left|\bigcup_{i'=i+1}^{i+k+1} M(i')\left(p\right)\right| \leq \ell \cdot \frac{1}{r^2} \cdot \frac{k}{32}
\]
marked children. We now delete from the solution all the marked paths obtaining a solution \(Q'\). Note that the source remains as it is never marked. This means that any sampled open path retains at least \(k/32\) children. For any path \(p\), we denote by \(N(p, r)\) the number of its descendants at distance \(r\) that have congestion induced by \(p\) bigger than \(2^9r\). Then if \(p\) is never marked at any round, we have that
\[
N(p, r) \leq \frac{1}{\log n} \left(\frac{k}{32}\right)^{\ell}.
\]

Indeed, if \(p\) belongs to layer \(i\) then \(\text{cong}(p, Q(i)) = 1/\gamma(p) \cdot \text{cong}(p, v_0) \leq 2\) by Constraint (\(7\)). Hence the first time we instantaneously bad events involving local congestion induced by the path \(p\), it is the case that none of its descendants are bad. Then if \(p\) is never marked we have that
\[
N(i, j)\left(p, r'\right) \leq \frac{\log n}{\log n} \left(\frac{k}{32}\right)^{\ell}
\]
at every iteration \(i \leq r' \leq i + \ell\), which means that few of the good descendants become bad at each round. Hence we obtain that
\[
N(p, r') \leq \frac{\log n}{\log n} \left(\frac{k}{32}\right)^{\ell}.
\]

This ensures that any remaining path has few bad descendants (in terms of local congestion).

We now define a set of paths \(R \subseteq Q\), which contains all paths with high local congestion. More precisely, let \(R\) contain for every path \(p \in Q\) and \(r' \leq r\) all paths \(p \in Q(p, r')\) such that \(\text{cong}(p, Q) > 2^9r^2\). Since we have no more marked paths, each \(p\) has at most \((\log n)^{-1} \cdot (k/32)^{\ell} \leq 1/(8\ell^2) \cdot (k/32)^{\ell}\) descendants at distance \(r'\) in \(R\). Applying Lemma 4 we can remove \(R\) and obtain a solution \(Q' \subseteq Q \setminus R\), where each open path has \(k/(64\ell)\) children and the source remains in the solution. Moreover, the solution has no more paths with \(\ell\)-local congestion more than \(2^{10}\ell^2\) and the global congestion is at most \(2^{11}\log^3 n\). This finishes the proof of Lemma 7.

**5 FROM SINGLE SOURCE TO MULTIPLE SOURCES**

We devise an algorithm that solves the problem with multiple sources based on an algorithmic technique first introduced by Haxell in the context of hypergraph matchings and then applied to a range of other problems, including the restricted assignment version of the Santa Claus problem. Haxell’s algorithmic technique can be thought of as a (highly non-trivial) generalization of the augmenting path algorithm for bipartite matching. Our algorithm makes only calls to the \(\alpha\)-approximation for a single source. The algorithm itself requires quasi-polynomial time as well (even if the \(\alpha\)-approximation ran in polynomial time). Our concrete variant relies on the following simple, but powerful subroutine.

**Lemma 24.** Let \(Q\) be a (single) degree-\(k'\) arborescence in a layered graph. Let \(R \subseteq Q\) be a set of paths where at most \((k'/4\ell)^i\) many end in each layer \(i\). Then there is a degree-\(k'/4\) arborescence \(Q' \subseteq Q \setminus R\). Furthermore, \(Q'\) can be computed in polynomial time.

**Proof.** We start pruning the arborescence from bottom to top and we maintain at every layer \(i\) that we remove at most \((k'/2\ell)^i\) paths in it. At the last layer \(i = h\) we remove simply the paths that are in \(R\). These are by definition at most \((k'/4\ell)^i \leq (k'/2\ell)^i\) many. Then assume we already pruned layers \(i, i + 1, \ldots, h\) and that we removed at most \((k'/2\ell)^i\) many paths in layer \(i\). Now in layer \(i - 1\) we again remove the at most \((k'/4\ell)^i - 1\) many paths in \(R\), but also those open paths where more than \(3/4 \cdot k'\) many children were removed in layer \(i\). The number of open paths removed this way is at most \((k'/2\ell)^i/(3/4 \cdot k') \leq 2/3 \cdot (k'/2\ell)^i - 1 < (k'/2\ell)^i - 1\). Clearly, this procedure maintains that every remaining open path has at least \(k'/4\) children and the source never gets removed. \(\square\)

Throughout the section we assume that \(k\) is the optimum, which is obtained through a binary search framework. We now assume that we have a degree-\(k/256a\) solution that already covers all but one source \(s_0 \in S\). We will augment this solution to one that covers all sources. This is without loss of generality, since we can apply the procedure iteratively \(|S|\) times, adding one source at a time.

**Blocking Trees and Addable Trees.** Let \(Q\) be our current solution (which does not cover \(s_0\)). On an intuitive level, blocking trees are \(k/256a\)-degree arborescences in our current solution \(Q\), which we would like to remove from the solution. In order to remove them, we need to add other arborescences for their sources instead. The addable trees are \(k/32a\)-degree arborescences not in our solution. Their sources are sources of blocking trees in \(Q\) and we would like to add to them solution. The addable trees in turn may be blocked by blocking trees, which means they overlap on some non-source vertex with a tree in \(Q\), preventing us from adding them to the solution. We note that addable trees are (by a factor of 8) better arborescences than what we ultimately need. As explained above, addable and blocking trees naturally form an alternating structure,
which is usually referred to as layers. In order to avoid conflicts with our other notion of layers, we call these rings. The addable and blocking trees in the ith ring will be denoted by \( A_i \) and \( B_i \).

**The Algorithm.** Initially there are no rings. We first run our \( \alpha \)-approximation to find a \( k/\alpha \)-degree arborescence for \( s_0 \) (without taking into account our current solution \( Q \)). We reduce this to a \( k/32\alpha \)-degree arborescence and store it as singleton set in \( A_1 \). Then we store in \( B_1 \) all arborescences in \( Q \) that intersect on any vertex with it. If the total intersection on each layer \( j \) is at most \( (k/128\alpha)^j \), then we can find through Lemma 24 a \( k/128\alpha \)-degree arborescence for \( s_0 \) which is disjoint from \( Q \). We reduce it to a \( k/256\alpha \)-degree arborescence and add it to the solution and terminate. Otherwise we continue the algorithm, but now we indeed have one ring of addable and blocking trees.

Assume now the algorithm in its current state has the rings \( A_1, B_1, \ldots, A_i, B_i \). Then we initialize \( A_{i+1} = \emptyset \) and add addable trees to it in the following Greedy manner. From our (layered) graph we produce a reduced instance by removing any vertices appearing in \( A_1, B_1, \ldots, A_i, B_i, A_{i+1} \) (except for the sources). Then we iterate over any source \( s \) that is either \( s_0 \) or the source of a blocking tree in \( B_1 \cup \cdots \cup B_i \). We call our \( \alpha \)-approximation for \( s \) on the reduced instance. If it outputs a \( k' \)-degree solution with \( k' \geq k/32\alpha \), we add it to \( A_{i+1} \) (after reducing to degree exactly \( k/32\alpha \)). We repeat until we can no longer add any addable trees to \( A_{i+1} \). Then we construct \( B_{i+1} \) by taking any arborescence in \( Q \) that intersects with any addable tree in \( A_{i+1} \). As before, we check if any addable tree in \( A_{i+1} \) has a total intersection of at most \( (k/128\alpha)^{j} \) on each layer \( j \) with solution \( Q \). If so, we collapse: we can compute through Lemma 24 a \( k/128\alpha \)-degree arborescence for the corresponding source \( s \), which is disjoint from \( Q \). We reduce it to a \( k/256\alpha \)-degree arborescence and add it to \( Q \). Now \( s \) has two arborescences in \( Q \). The other arborescence must have appeared as a blocking tree in an earlier ring \( B_j \). We remove this blocking tree from \( Q \) and delete all rings after \( j \). Then we revisit the addable trees in \( A_j \). Since we removed a blocking tree in \( B_j \), one of them may now have a small intersection with all trees in \( Q \) (as above). If so, we collapse this as well. We continue until no more collapse is possible, leaving us with a new solution \( Q \) and a prefix of the previous rings. We then continue the algorithm. Once an addable tree for \( s_0 \) is added to \( Q \) we terminate.

For the analysis of the algorithm we refer to the full version of the paper.

**REFERENCES**

[1] Chidambaram Annamalai. 2019. Lazy local search meets machine scheduling. *SIAM J. Comput.*, 48, 5 (2019), 1503–1543.

[2] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. 2017. Combinatorial algorithm for restricted max-min fair allocation. *ACM Transactions on Algorithms* 13, 3 (2017), 1–28.

[3] Arash Asadpour, Uriel Feige, and Amin Saberi. 2012. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms* 8, 3 (2012), 24:1–24:9.

[4] Étienne Bamas, Paritosh Garg, and Lars Rohwedder. 2021. The Submodular Santa Claus Problem in the Restricted Assignment Case. In *Proceedings of ICALP*.

[5] Nikhil Bansal. 2017. Scheduling: Open problems old and new. Presentation at MAPSP.

[6] Nikhil Bansal and Maxim Sviridenko. 2006. The santa claus problem. In *Proceedings of STOC*. 31–40.

[7] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. 2009. Maxmin allocation via degree lower-bounded arborescences. In *Proceedings of STOC*. 543–552.

[8] Ivona Bezáková and Varsha Dani. 2005. Allocating indivisible goods. *ACM SIGecom Exchange* 5, 3 (2005), 11–18.

[9] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. 2009. On allocating goods to maximize fairness. In *Proceedings of FOCS*. 107–116.

[10] Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. 2014. On (1,\( \varepsilon \))-Restricted Assignment Makespan Minimization. In *Proceedings of SODA*. 1087–1101.

[11] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. 1999. Approximation algorithms for directed Steiner problems. *Journal of Algorithms* 33, 1 (1999), 73–91.

[12] Siu-Wing Cheng and Yucheng Mao. 2018. Restricted Max-Min Fair Allocation. In *Proceedings of ICALP*. Vol. 107. 37:1–37:13.

[13] Siu-Wing Cheng and Yuchen Mao. 2019. Restricted Max-Min Allocation: Approximation and Inequality Gap. In *Proceedings of ICALP*. 38:1–38:13.

[14] Herman Chernoff. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. The Annals of Mathematical Statistics 25, 2, 595–577.

[15] Sami Davies, Thomas Rothvoß, and Yihao Zhang. 2020. A tale of Santa Claus, hypergraphs and matroids. In *Proceedings of SODA*. 2748–2757.

[16] Tomás Ebenlendr, Marek Krčál, and Jiří Spališ. 2014. Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines. *Algorithmica* 68, 1 (2014), 62–80.

[17] Uriel Feige. 2008. On allocations that maximize fairness. In *Proceedings of SODA*. 287–299.

[18] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Arvind Sinhaivasan, and Nan Wang. 2007. Inequality ratio for group Steiner trees and directed Steiner trees. *SIAM J. Comput.*, 36, 5 (2007), 1494–1511.

[19] Eran Halperin and Robert Krauthgamer. 2003. Polylogarithmic inapproximability. In *Proceedings of STOC*. 585–594.

[20] Penny Haxell and Tibor Szabó. 2023. Improved Inequality Gap in Max-Min Allocation: or Topology at the North Pole. In *Proceedings of SODA*. to appear.

[21] Penny H. Haxell. 1995. A condition for matchability in hypergraphs. *Journal of Combinatorial Theory, Series B* 64, 1 (1995), 254–248.

[22] Klaus Jansen, Kati Land, and Marten Maack. 2018. Estimating the makespan of the two-valued restricted assignment problem. *Algorithmica* 80, 4 (2018), 1357–1382.

[23] Klaus Jansen and Lars Rohwedder. 2019. Local Search Breaks 1.75 for Graph Balancing. In *Proceedings of ICALP*. 741–7414.

[24] Klaus Jansen and Lars Rohwedder. 2020. A quasi-polynomial approximation for the restricted assignment problem. *SIAM J. Comput.* 49, 6 (2020), 1083–1108.

[25] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming* 46, 1 (1990), 259–271.

[26] Robin A Moster and Gábor Tardos. 2010. A constructive proof of the general Lovász local lemma. *Journal of the ACM (JACM)* 57, 2 (2010), 1–15.

[27] Lukáš Poláček and Ola Svensson. 2015. Quasi-polynomial local search for restricted max-min fair allocation. *ACM Transactions on Algorithms* 12, 2 (2015), 5:1–5:13.

[28] Petra Schuurman and Gerhard J. Woeginger. 1999. Polynomial-time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2, 5 (1999), 203–213.

[29] Ola Svensson. 2012. Santa Claus Schedules Jobs on Unrelated Machines. *SIAM J. Comput.*, 41, 5 (2012), 1318.

[30] José Verschae and Andreas Wiese. 2014. On the configuration-LP for scheduling on unrelated machines. *Journal of Scheduling* 17, 4 (2014), 371–383.

[31] David P. Williamson and David B. Shmoys. 2011. *The design of approximation algorithms*. Cambridge university press.