Coordination Mechanisms for Selfish Routing over Time on a Tree

Sayan Bhattacharya[1], Janardhan Kulkarni[2⋆], Vahab Mirrokni[3]

[1] Max-Planck Institute for Informatics, Saarbrucken, Germany.
[2] Duke University, Durham, USA.
[3] Google Inc., New York, USA

Abstract. While selfish routing has been studied extensively, the problem of designing better *coordination mechanisms* for routing over time in general graphs has remained an open problem. In this paper, we focus on tree networks (single source multiple destinations) with the goal of minimizing (weighted) average *sojourn time* of jobs, and provide the first coordination mechanisms with provable price of anarchy for this problem. Interestingly, we achieve our price of anarchy results using simple and strongly local policies such as Shortest Job First and the Smith's Rule (also called HDF). In particular, for the case of unweighted jobs, we design a coordination mechanism with polylogarithmic price of anarchy. For weighted jobs, on the other hand, we show that price of anarchy is a function of the depth of the tree and accompany this result by a lower bound for the price of anarchy for the Smith Rule policy and other common strongly local scheduling policies.

Our price of anarchy results also imply improved approximation algorithms for the underlying optimization problem of routing over a tree. This problem is well motivated from applications of routing in supercomputers and data center networks where average sojourn time is an important metric.

1 Introduction

As a central topic in algorithmic game theory, selfish routing problems have been studied extensively in the context of *congestion games* [20, 26, 27]. Being a representative class of potential games, network congestion games have served as a foundation for proving price of anarchy results. However they lack an important aspect of real network routing which is the fact that *routing happens over time*, and any realistic model should take this into account. To address this issue, several new models have been proposed to capture the nature of realistic routing over time [10, 11, 21, 19, 24, 13, 12, 4]. Amongst these models, the concept of *coordination mechanisms*, first introduced in an influential paper by Christodoulou, Koutsoupias, and Nanavati [10], have been proposed to capture the queueing nature of routing. Coordination mechanisms model the decentralized nature of routing decisions made by machines and the selfish behavior of jobs: they do so by seeking local policies that achieve a good price of anarchy in the resulting equilibria in a corresponding game. While these subjects have attracted a great amount

⋆ Supported by NSF Awards CCF-1008065 and IIS-0964560.

of research, the problem of designing better coordination mechanisms for routing over time has remained a wide open problem, and results have been developed only for very special classes of networks such as parallel edges (corresponding to a multi-processor scheduling problem [10, 17, 12, 4]). In this paper, we focus on tree networks, and provide the first coordination mechanisms with provable price of anarchy.

The significance of our results are two-fold: other than providing approximately optimal coordination mechanisms, our price of anarchy results also imply improved approximation algorithms for the underlying optimization problem of routing over a tree. This problem is well motivated from applications of routing in supercomputers and data centers where average sojourn time is an important QoS metric. Before elaborating the model, we describe the significance of the optimization problem over trees in various applications.

Applications of optimizing routing over a tree: In 1985, Leiserson discovered that a variant of tree topology called *fat tree network* is a *universal network* for efficient network connections [23]. In a typical fat tree topology, each tree edge has a capacity (or bandwidth or processing power) and the edges closer to the root node have higher capacities than those to the leaves. The fat tree topology quickly became the de facto standard for connecting processors within supercomputers. Recently, the efficiency of tree topologies is being rediscovered in the context of data center architectures as data center sizes grow exponentially with the explosion of the cloud-based services. A typical server farm in a data center consists of a set of web severs interconnected by a fat tree topology. See, for example, an important paper on this topic by Al-Fares, Loukissas and Vahdat [1]. Despite such important applications of optimal routing tasks over tree topologies, very little is known from a theoretical standpoint about scheduling jobs over a tree network to minimize their *average sojourn time* which is a fundamental quality of service metric for evaluating the performance of such a system. The sojourn time of a job is defined as the total time the job spends in the network. On the other hand, minimizing the *makespan* of a schedule, defined as the maximum sojourn time over all the jobs, has a vast literature in these settings [14, 25, 24], including the celebrated result of Leighton, Maggs and Rao [22].

1.1 The Model

We are given a tree $T = (V, E)$ rooted at node $r \in V$. Each edge $e \in E$ in the tree is a machine[4] with *speed* $s(e)$. For the rest of the paper, we use the terms "edge" and "machine" interchangeably. There is a set of jobs J with unique identifiers, which will be used by our policies for breaking ties consistently. Each job $j \in J$ has *weight* w_j and *length* p_j, and its *processing time* on edge e is $p_{je} = p_j/s(e)$. Each edge can process at most one unit of the jobs during a unit time-step, and a job j requires p_{je} time-units of processing on an edge e. At time $t = 0$, all the jobs are located at the root.

A job j can be *served* by only the nodes in the subset $\mathcal{L}_j \subseteq V$. This models, for example, the fact that some servers in a data center may not have the necessary content to satisfy a request. The job j starts at the root and wants to exit the tree through any

[4] The reason that we use the term "machine" to refer to the "edges" is to cope with the scheduling literature. One should think about these machines as a communication link.

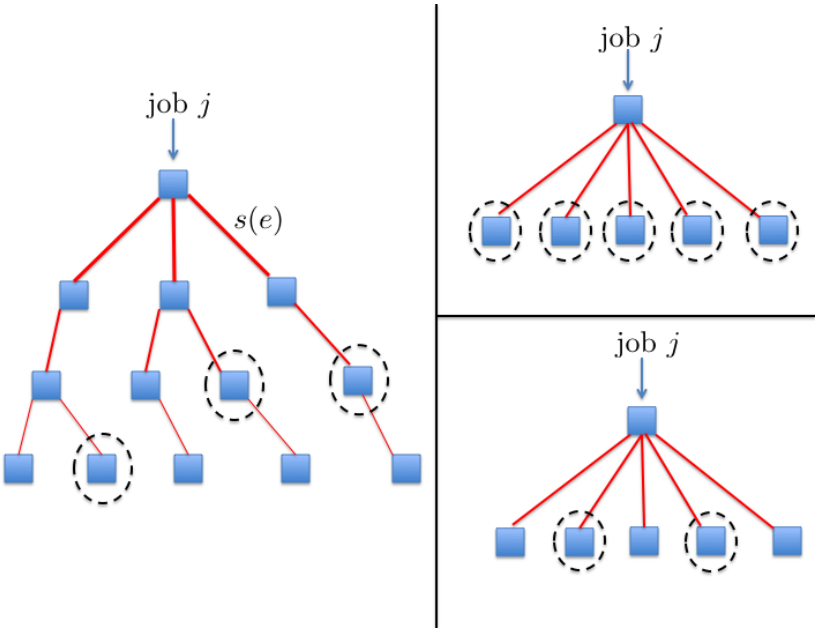

Fig. 1. In our model, each edge e has a speed $s(e)$. Each job j starts at the root and needs to travel to one of the nodes in $\mathcal{L}_j$, shown in dashed circles. To the right, we show how our model generalizes the related machines (top right) and the restricted assignment setting (bottom right).

one of the nodes in $\mathcal{L}_j$, which are called the *destination-nodes* of j. Accordingly, the job j selects a path $i = (e_1, \ldots, e_l)$ that begins at the root of the tree and terminates at some node in $\mathcal{L}_j$. Here, e_1 is the first edge on the path (adjacent to the root), and e_l is the last edge. The job can start getting processed on an edge e_k, $k \in \{2, \ldots, l\}$, only after it is processed completely by the preceding edge e_{k-1}. The job exits the tree when it gets completely processed on the last edge e_l, and the time at which this event takes place is called the *sojourn time* of the job. The weighted sojourn time of j is equal to its weight w_j times its sojourn time. Note that since all the jobs are at the root node at time $t = 0$, the average sojourn time is equivalent to the average *flow-time* in our context. A reader familiar with the scheduling literature can see that our model is a generalization of the *related machine* and the *restricted assignment* settings (see Figure 1).

The *underlying optimization problem* asks us to route every job j through a root-to-destination path terminating in $\mathcal{L}_j$, and provide a scheduling policy on each edge so as to minimize the sum of their weighted sojourn times. We allow preemption of jobs on an edge. The jobs, however, are selfish agents who cannot be forced to obey the dictate of a centralized authority. Thus, in our model, the machines declare their scheduling policies in advance, and this induces a simultaneous-move game between the jobs. We require that the scheduling policies be *strongly local*, in the sense that the scheduling decision by an edge at any time-instant depends only on the current set of jobs waiting to be processed on that edge, and is independent of the global state of the system.

In this game, the strategy of each job j consists of selecting a path from the root to any one of the destination-nodes in $\mathcal{L}_j$. The vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{|J|})$ denotes an outcome (strategy-profile) of the game, where $\boldsymbol{\theta}_j$ is the path selected by the job j. The symbol $(i, \boldsymbol{\theta}_{-j})$ denotes an outcome where the job j selects the path i, and every job $j' \neq j$ selects the path $\boldsymbol{\theta}_{j'}$. The symbol $\text{Cost}_j(\boldsymbol{\theta})$ denotes the cost incurred by the job j under the outcome $\boldsymbol{\theta}$, which is equal to its weighted sojourn time. An outcome $\boldsymbol{\theta}$ is in a (pure) Nash equilibrium iff no job can reduce its cost by unilaterally deviating to another path, i.e., iff $\text{Cost}_j(\boldsymbol{\theta}) \leq \text{Cost}_j(i, \boldsymbol{\theta}_{-j})$ for all jobs j and root-to-destination paths i terminating at a node in $\mathcal{L}_j$.

The *price of anarchy* (PoA) of the game is the worst (maximum) possible ratio between the total cost of the agents in a Nash equilibrium and the optimal objective of the underlying optimization problem. Intuitively, it is a measure of the degradation in the overall system-performance due to the strategic interactions between the jobs.

We want to solve the following problem: Find a set of scheduling policies for the machines so as to minimize the PoA of the resulting game.

1.2 Our Contributions and Techniques

We analyze the PoA of the game induced among the jobs when the machines follow a natural and easy to implement scheduling policy known as Smith's Rule [28] or "Highest Density First" (HDF). Under this policy, at every time-step a machine works on a job j that is available for processing and has maximum "density" w_j/p_j. When all jobs have the same weight, this reduces to the "Shortest Job First" (SJF) scheduling policy. If multiple jobs happen to have the same density, then *all the machines use the same tie-breaking rule* to decide which one of these jobs to consider first. Hence, without loss generality, we will assume throughout the rest of the paper that no two jobs have exactly the same density.

Unweighted Jobs. When the jobs are unweighted and the machines follow SJF policy, we prove that the PoA of the induced game is $O(\log^2(s_{\max}/s_{\min}))$ (Theorem 3). Here, $s_{\max}$ (resp. $s_{\min}$) denotes the speed of the fastest (resp. slowest) machine. Note that this implies constant PoA when all the machines are identical; even in the general case, the PoA is independent of the number of jobs or nodes in the network.

Weighted Jobs. When different jobs have different weights and all the edges follow the Smith Rule (HDF) policy, we prove that the PoA is $O(d^2)$, where d is the depth of the tree (Theorem 5). We complement this result by showing that well-known scheduling policies – such as HDF, Weighted Round Robin, or Weighted Shortest Elapsed Time First – lead to games whose PoA depend polynomially on d (Theorems 4, 7).

Approximation Algorithms. It is known that Smith's Rule (HDF) induces a game with a pure NE [15] which can be computed in polynomial time by greedily adding jobs in the decreasing order of their densities. This implies that our PoA results also lead to approximation algorithms for the underlying optimization problems (both for the weighted and unweighted jobs), the approximation ratio being the same as the PoA of the game.

Our Technique: Both of the PoA upper-bounds in this paper are obtained using the following simple technique: First, we find an LP relaxation for the underlying optimization problem and write down its dual. Next, we consider any arbitrary Nash equilibrium outcome $\boldsymbol{\theta}$, and based on this outcome assign values to the dual variables in such a way that satisfies all the dual constraints, thereby getting a feasible solution to the dual LP. Finally, we show that the objective of this feasible dual solution is at least $1/\alpha$ times the total cost incurred by the jobs under $\boldsymbol{\theta}$. Weak duality implies that the PoA of the game is at most α. Our overall approach is inspired by papers [2, 6]. This technique is very powerful and can potentially be applied to bound the PoA in several other settings. Bilò et al. [7] give another application of this technique to analyze PoA.

Apart from the overall idea of using the dual fitting technique to analyze the PoA of the game, writing a linear programming relaxation with small integrality gap turns be a significant challenge for our problem. A direct extension of the time-indexed LP [2] has a huge integrality gap in our setting. We circumvent this difficulty for the case of unweighted jobs by first finding a set of *critical edges* which play a crucial role in how the jobs delay each other. Then, we write a time-indexed LP relaxation taking into account only these edges which brings the integrality gap down. See Section 2 for details.

Related Work. Following the landmark paper of Christodoulou et al. [10] who initiated the study of coordination mechanisms, several papers have been written on the topic for various problems. However, most of these results are for machine scheduling problems, either proving PoA results on the makespan objective function [17, 8, 4] or recently for the weighted completion times [12, 6]. In the context of selfish routing, the multi-machine scheduling problem corresponds to a network of parallel edges and related machine scheduling is a special case of our model where the tree is a star, i.e, a tree of depth one (Figure 1). The only two results that go beyond the scheduling problem are by Hoefer et al [15] and by Christodoulou et al. [11]. The first paper only studies existence and computation of equilibria for various coordination mechanisms and leaves the PoA question as an open problem. The second paper discusses a quite different setting with non-atomic players.

Our work is also related to the literature on the PoA of selfish routing [20, 27], and more specifically unsplittable selfish routing [26]. Here we extend the selfish routing model by incorporating a temporal component into the problem formulation. Other attempts to address this issue include [21, 19, 13, 24], but none of these results discuss coordination mechanisms using strongly local (decentralized) policies.

We have recently learned through personal communication that scheduling over tree and line networks are also being considered in the online (and resource augmentation) setting in a recent (ongoing) work by Im et al. [16] and Antoniadis et al. [3]. Finally, our work is related to approximation algorithms of classical optimization problems for minimizing the weighted sum of completion times and flow times [18, 9, 2, 5], none of which present an approximation algorithm for the problem minimizing average completion time in routing over a tree.

2 $O(\log^2(s_{\max}/s_{\min}))$ PoA for Unweighted jobs

In this section, we assume that all the jobs have unit weights, and bound the PoA of the game where every edge follows the Shortest Job First (SJF) scheduling policy (Theorem 3). We start with a high-level overview of our approach. We say that a job j is *delayed* by another job j' on an edge e at time t iff two conditions are satisfied at the same time: (a) the job j is available for processing on edge e, and (b) instead of j, the edge is processing the job j'. A machine following the SJF policy never sits idle when one or more jobs are waiting in its queue. This ensures that the sojourn time of a job j is exactly equal to the total amount of processing done on j by all the edges, plus the total delay it encounters due to all other jobs. The former quantity is given by $\sum_{e \in i} p_{je}$, where i is the path selected by the job. It is the latter quantity for which it is difficult to get a closed-form expression. We overcome this difficulty by showing that there is a small subset of *critical edges* on any path (Definition 1), and that one job can delay another only on one of these edges. This line of reasoning culminates in a key structural result (Theorem 2) that gives an upper bound on the maximum delay a job j can experience due to any single job $j' \neq j$.

As alluded in Section 1.2, we now confront the task of designing a suitable LP relaxation for the underlying optimization problem. A straightforward extension of the time-indexed LP considered in [2] to our setting leads to a large integrality gap. On the positive side, the time-indexed LP has one nice property: Its dual has variables that can be interpreted as decomposing the total delay incurred by a job across the edges on its path. The duals of other "natural" LP relaxations for our problem do not seem to be amenable to such a nice interpretation. Accordingly, we modify the time-indexed LP relaxation by only taking into account the critical edges of the tree. As the number of critical edges is small, this brings down the integrality gap of the LP. We then manage to fit its dual using Theorem 2.

To proceed with the technical details, let $s_{\max}$ (resp. $s_{\min}$) denote the speed of the fastest (resp. slowest) of a machine, and let $K = \lfloor \log(s_{\max}/s_{\min}) \rfloor$. For ease of exposition, we assume that the speeds of the machines are discretized in powers of two. By standard *time stretching* arguments, it is easy to show that this assumption can lead to at most a factor two loss in the PoA of our coordination mechanism (see Appendix A).

Assumption 1 *For all $e \in E$, we have $s(e) = 2^k \cdot s_{\min}$ for some $k \in \{1, 2, \ldots, K\}$.*

Definition 1. *Let $e_{i,k}$ denote the edge of minimum depth on path i that has speed $2^k s_{\min}$. We refer to such an edge as a* critical *edge on that path. We define $E_i = \cup_k \{e_{ik}\}$ to be the set of all critical edges on path i.*

Below, we describe our LP relaxation for the underlying optimization problem. For rest of the paper, we overload the notation $i \in \mathcal{L}_j$ to denote a path i that starts at the root and terminates at a node in $\mathcal{L}_j$.

$$\text{Minimize} \qquad \sum_j \sum_{i \in \mathcal{L}_j} \sum_{e \in E_i} \sum_t x_{ijet} \cdot \left(\frac{t}{p_{je}} \right) + \sum_j \sum_{i \in \mathcal{L}_j} \sum_e \sum_t x_{ijet} \qquad (1)$$

$$\sum_{i \in \mathcal{L}_j} x_{ij} \geq 1 \qquad \forall \text{ jobs } j \tag{2}$$

$$\sum_{t} \frac{x_{ijet}}{p_{je}} \geq x_{ij} \qquad \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j, \text{ edges } e \in i \tag{3}$$

$$\sum_{j} \sum_{i \in \mathcal{L}_j : e \in i} x_{ijet} \leq 1 \qquad \forall \text{ edges } e, \text{ times } t \tag{4}$$

$$x_{ijet}, x_{ij} \geq 0 \qquad \forall j, i \in \mathcal{L}_j, e \in i, t \tag{5}$$

In an integral feasible solution of the above linear program, the variable $x_{ij} \in \{0, 1\}$ indicates if the job j takes the path $i \in \mathcal{L}_j$. The variable $x_{ijet} \in \{0, 1\}$ indicates if the job j takes the path $i \in \mathcal{L}_j$ and is being processed on the edge $e \in i$ at time-step t.

Constraint 2 states that every job has to take some path. Constraint 3 states that if a job j takes a path $i \in \mathcal{L}_j$, then it has to get completely processed on all the edges on this path. Finally, constraint 4 states that every edge can process at most one unit of the jobs during one time-step.

The second summation in the LP objective gives the total amount of processing done on all the jobs, which clearly is a lower bound on the sum of their sojourn-times. Now, fix any job j which takes a path $i \in \mathcal{L}_j$, and consider an edge $e \in i$ on this path. The term $\sum_t x_{ijet} \cdot (t/p_{je})$ is known as the *fractional completion time* [2] of the job j on the edge e. This is at most the time at which the edge e finishes processing the job, which, in turn, is at most the sojourn-time of j. We sum this quantity over all the critical edges E_i on path i, and the number of such critical edges is $O(K)$. Thus, the summation $\sum_{i \in \mathcal{L}_j} \sum_{e \in E_i} \sum_t (t/p_{je}) \cdot x_{ijet}$ is $O(K)$ times the sojourn-time of j. Summing over all the jobs, we see that the overall LP objective is $O(K)$ times the objective of the underlying optimization problem.

We now get a new LP by replacing the 1 in the right hand side of constraint 4 with $1/(4K)$. This imposes the condition that a machine can process at most $1/(4K)$ units of the jobs during one time-step, and, by standard scaling arguments, it is easy to show that this increases the LP objective by a factor of $4K$ (see Lemma 12 in Section D.1). The dual of this new LP is given by LP (6). By weak duality, we get Lemma 1.

$$\text{Max} \qquad \sum_{j} y_j - \frac{1}{4K} \sum_{e,t} z_{et} \tag{6}$$

$$y_j \leq \sum_{e \in i} u_{ije} \qquad \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j \tag{7}$$

$$\frac{u_{ije}}{p_{je}} - z_{et} \leq 1 \qquad \forall \text{ jobs } j, i \in \mathcal{L}_j, e \in i \setminus E_i, \text{ times } t \tag{8}$$

$$\frac{u_{ije}}{p_{je}} - z_{et} \leq \frac{t}{p_{je}} + 1 \qquad \forall \text{ jobs } j, i \in \mathcal{L}_j, e \in E_i, \text{ times } t \tag{9}$$

$$y_j, u_{ije}, z_{et} \geq 0 \qquad \forall j, t, i \in \mathcal{L}_j, e \in i \tag{10}$$

Lemma 1. *The objective of any feasible solution to LP (6) is $O(K^2)$ times the optimal objective of the underlying optimization problem, where $K = \lfloor \log(s_{\max}/s_{\min}) \rfloor$.*

For the rest of this section, we will assume that every edge follows SJF scheduling policy, and we will analyze the PoA of the resulting game. The theorem below bounds the maximum delay a job can encounter due to any other job.

Theorem 2. *Suppose that every machine runs SJF scheduling policy. Consider any two jobs j^*, j_1^*, and fix any outcome $\boldsymbol{\theta}$ (not necessarily a Nash Equilibrium) of the induced game. Let $e \in \boldsymbol{\theta}_{j^*} \cap \boldsymbol{\theta}_{j_1^*}$ be an edge with slowest speed that is common to the paths taken by the jobs. Then the total delay of the job j^* due to the job j_1^* is at most $2p_{j^*}/s(e)$.*

Proof (Sketch). Consider the path $\boldsymbol{\theta}_{j^*} = (e_1, \ldots, e_l)$, where e_1 (resp. e_l) is the edge adjacent to (resp. farthest away from) the root. We decompose this path in w segments, for some natural number w, in the following manner. Consider a function $f : [1, w+1] \to [1, l+1]$ such that $1 = f(1) < f(2) < \cdots < f(w+1) = l+1$. Segment $k \in [1, w]$ corresponds to the sequence of edges $e_{f(k)}, e_{f(k)+1}, \ldots, e_{f(k+1)-1}$. The decomposition satisfies two properties.

- The speeds of the first edges of these segments form a *strictly* decreasing sequence. Hence, Assumption 1 implies that $s(e_{f(1)}) > s(e_{f(2)})/2 > \cdots > s(e_{f(w)})/2^{w-1}$.
- Within each segment, the speed of the first edge is at most the speed of any other edge. Thus, $s(e_{f(k)}) \leq s(e)$ for all $k \in [1, w]$ and $e \in \{e_{f(k)}, \ldots, e_{f(k+1)-1}\}$.

Note that the first edge $e_{f(k)}$ of every segment $k \in [1, w]$ is a critical edge on the path $\boldsymbol{\theta}_{j^*}$. We show that the job j^* can get delayed by other jobs *only* on this set of edges $\{e_{f(k)} : k \in [1, w]\}$. Now, under SJF policy, the job j^* can get delayed by another job j_1^* only if $p_{j_1^*} \leq p_{j^*}$, and, in this case, the delay experienced by j^* due to j_1^* on any edge $e_{f(k)}$ is at most $p_{j_1^*}/s(e_{f(k)}) \leq p_{j^*}/s(e_{f(k)})$. Thus, the total delay incurred by j^* due to j_1^* is upper bounded by the sum $\sum_{k=k^*}^{1} p_{j^*}/s(e_{f(k)})$, where k^* is the segment with the largest index whose first edge also belongs to $\boldsymbol{\theta}_{j_1^*}$. This sum is part of a geometric series with common ratio $1/2$, and is at most $2p_{j^*}/s(e_{f(k^*)})$. The theorem follows from the fact that the edge $e_{f(k^*)}$ has the slowest speed among all the edges that are common to the paths $\boldsymbol{\theta}_{j^*}$ and $\boldsymbol{\theta}_{j_1^*}$. See Appendix B for the complete proof.

In Figure 2, we give an algorithm that takes any Nash equilibrium of the game (under SJF policy), and, depending on this input, sets the variables in LP (6).

Lemma 2. *If the dual variables are assigned values as in Figure 2, then for all jobs j and root-to-leaf paths $i \in \mathcal{L}_j$, we have $\mathit{Cost}_j(i, \boldsymbol{\theta}_{-j}) = \sum_{e \in i} u_{ije}$.*

Proof. Fix the outcome $(i, \boldsymbol{\theta}_{-j})$. Now, the sojourn-time of j equals the amount of time j is processed, *plus* the amount of time it is delayed due to the other jobs. The former quantity is equal to $\sum_{e \in i} p_{je}$, the latter quantity being $\sum_{j' \neq j} \delta_{ij}(j')$. Thus, we get:

$$\begin{aligned} \mathrm{Cost}_j(i, \boldsymbol{\theta}_{-j}) = \sum_{e \in i} p_{je} + \sum_{j' \neq j} \delta_{ij}(j') &= \sum_{e \in i} p_{je} + \sum_{e \in E_i} \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j') \qquad (11) \\ &= \sum_{e \in i \setminus E_i} p_{je} + \sum_{e \in E_i} \left(p_{je} + \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j') \right) \\ &= \sum_{e \in i \setminus E_i} u_{ije} + \sum_{e \in E_i} u_{ije} = \sum_{e \in i} u_{ije} \end{aligned}$$

INPUT: Any outcome $\boldsymbol{\theta}$ of the game (under SJF policy) that is in a Nash equilibrium.

1. Set $y_j \leftarrow \text{Cost}_j(\boldsymbol{\theta})$.
2. Consider an indicator variable $\lambda_{jet} \in \{0, 1\}$ that is set to 1 iff
   (a) the path $\boldsymbol{\theta}_j$ contains the edge e as a critical edge, i.e., $e \in E_{\boldsymbol{\theta}_j}$ and
   (b) the sojourn-time of job j is at least t under the outcome $\boldsymbol{\theta}$.
   Set $z_{et} \leftarrow \sum_j 2 \cdot \lambda_{jet}$.
3. Set the variable u_{ije} as follows.
   (a) If $e \neq E_i$, then $u_{ije} \leftarrow p_{je}$.
   (b) Else if $e \in E_i$, then let Γ_{ije} denote the set of all jobs $j' \neq j$ such that e is the slowest edge in $i \cap \boldsymbol{\theta}_{j'}$. Let $\delta_{ij}(j')$ be the total delay experienced by j due to the job j' under the outcome $(i, \boldsymbol{\theta}_{-j})$. Set $u_{ije} \leftarrow p_{je} + \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j')$.

Fig. 2. Setting the dual variables in LP (6) for unweighted jobs.

To see why equation 11 holds, define J^+ to be the set of jobs which make positive contributions towards the sum $\sum_{j' \neq j} \delta_{ij}(j')$, i.e., $J^+ = \{j' \neq j \,:\, \delta_{ij}(j') > 0\}$. Each job $j' \in J^+$ has $i \cap \boldsymbol{\theta}_{j'} \neq \emptyset$, and there is a unique critical edge on path i that is also the slowest edge in $i \cap \boldsymbol{\theta}_{j'}$. So each job $j' \in J^+$ is part of *exactly one* of the sets in $\{\Gamma_{ije} : e \in E_i\}$. In other words, $\{\Gamma_{ije} \,:\, e \in E_i\}$ induces a partition of the jobs in J^+.

Lemma 3. *If the dual variables are assigned values as in Figure 2, then they satisfy the constraints 7 and 8 of LP (6).*

Proof. The right hand side of constraint 7, by Lemma 2, is equal to $\text{Cost}_j(i, \boldsymbol{\theta}_{-j})$. The left hand side, by Figure 2, is equal to $\text{Cost}_j(\boldsymbol{\theta})$. The constraint is satisfied as the Nash equilibrium condition dictates that $\text{Cost}_j(\boldsymbol{\theta}) \geq \text{Cost}_j(i, \boldsymbol{\theta}_{-j})$.

In constraint 8, the edge e is not a critical edge on the path i. Hence, by Figure 2, the quantity z_{et} is zero at all times t. Furthermore, the quantity u_{ije} is set to p_{je}, so that the left hand side of the constraint is 1, which is equal to its right hand side.

Lemma 4. *Fix any job j, any path $i \in \mathcal{L}_j$, any critical edge $e \in E_i$, and any job $j' \in \Gamma_{ije}$. Let $\delta_{ij}(j', t)$ be the total delay experienced by j due to the job j' (anywhere in the tree) on or after time t, under the outcome $(i, \boldsymbol{\theta}_{-j})$. We have $\delta_{ij}(j', t) \leq 2\lambda_{j'et} \cdot p_{je}$.*

Proof. The main difficulty in proving the lemma is that $\delta_{ij}(j', t)$ refers to the outcome $(i, \boldsymbol{\theta}_{-j})$, whereas $\lambda_{j'et}$ refers to the outcome $\boldsymbol{\theta}$. So we introduce the quantity $\lambda^*_{j'et}$, which is the exact analogue of $\lambda_{j'et}$ under the outcome $(i, \boldsymbol{\theta}_{-j})$. Note that the edge e already belongs to $\boldsymbol{\theta}_{j'}$ and is a critical edge. Hence, we drop condition (a) used in defining $\lambda_{j'et}$ in Figure 2. We then consider two possible cases.

$$\lambda^*_{j'et} = \begin{cases} 1 & \text{if the sojourn-time of } j' \text{ under } (i, \boldsymbol{\theta}_{-j}) \text{ is at least } t; \\ 0 & \text{otherwise.} \end{cases}$$

By our assumption in the first paragraph of Section 1.2, either $p_j < p_{j'}$ or $p_j > p_{j'}$.

Case 1. Either $\lambda^*_{j'et} = 0$ or $p_j < p_{j'}$. Here, if $\lambda^*_{j'et} = 1$, then the job j' is already *out of the system* by time t under the outcome $(i, \boldsymbol{\theta}_{-j})$. Naturally, we have $\delta_{ij}(j', t) = 0$, and

the lemma holds. On the other hand, if $p_j < p_{j'}$, then SJF scheduling policy ensures that the job j never gets delayed by the job j', and we again have $\delta_{ij}(j', t) = 0$.

Case 2. $\lambda^*_{j'et} = 1$ and $p_j > p_{j'}$. In this case, first note that by Theorem 2, $\delta_{ij}(j', t) \leq 2p_{je}$. Since $\lambda^*_{j'et} = 1$, we get:

$$\delta_{ij}(j', t) \leq 2\lambda^*_{j'et} \cdot p_{je} \tag{12}$$

Since $p_j > p_{j'}$, SJF scheduling policy ensures that the processing of j' is not affected if j switches its path. Specifically, the time-steps at which j' is processed by edge e remains unchanged under the two outcomes $(i, \boldsymbol{\theta}_{-j})$ and $\boldsymbol{\theta}$. Thus, we have $\lambda_{j'et} = \lambda^*_{j'et}$, and equation 12 implies that the lemma holds.

Lemma 5. *If the dual variables are assigned values as in Figure 2, then they satisfy all the constraints of LP (6).*

Proof. By Lemma 3, the constraints 7, 8 are already satisfied. We focus on the remaining constraint 9. Fix any job j, any path $i \in \mathcal{L}_j$, and any edge $e \in E_i$. By Lemma 4:

$$\sum_{j' \in \Gamma_{je}} \delta_{ij}(j', t) \leq p_{je} \cdot \sum_{j' \in \Gamma_{je}} 2\lambda_{j'et} \leq p_{je} \cdot z_{et} \tag{13}$$

Under the outcome $(i, \boldsymbol{\theta}_{-j})$, the total delay experienced by j due to the jobs in Γ_{je} till time-step t is, by definition, at most t. This leads to the following inequality.

$$\sum_{j' \in \Gamma_{je}} (\delta_{ij}(j') - \delta_{ij}(j', t)) \leq t \tag{14}$$

From Figure 2, equation 14 and equation 13, we see that constraint 9 is satisfied.

$$u_{ije} = p_{je} + \sum_{j' \in \Gamma_{je}} \delta_{ij}(j') \leq p_{je} + t + \sum_{j' \in \Gamma_{je}} \delta_{ij}(j', t) \leq p_{je} + t + p_{je} \cdot z_{et}.$$

Lemma 6. *If the dual variables are set as in Figure 2, then the objective of LP (6) is at least* $(1/2) \cdot \sum_j$ *Cost*$_j(\boldsymbol{\theta})$.

Proof. Fix the outcome $\boldsymbol{\theta}$, and focus on any job j with sojourn-time $\text{Cost}_j(\boldsymbol{\theta})$.

(Case 1) $t \leq \text{Cost}_j(\boldsymbol{\theta})$. In this case, Figure 2 implies that the job j contributes 2 to each of the z_{et}'s corresponding to the critical edges in the path $\boldsymbol{\theta}_j$, and it makes zero contribution to the remaining z_{et}'s. Since $\boldsymbol{\theta}_j$ has at most K critical edges, the total contribution of the job to the sum $\sum_e z_{et}$ is at most $2K$.

(Case 2) $t > \text{Cost}_j(\boldsymbol{\theta})$. Here, by Figure 2, the job j contributes 0 to the sum $\sum_e z_{et}$.

Summing over all time-steps, the contribution of any single job j to the sum $\sum_{e,t} z_{et}$ is at most $2K \cdot \text{Cost}_j(\boldsymbol{\theta})$. Next, summing over the contributions from all the jobs, we infer that $\sum_{et} z_{et} \leq (2K) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta})$. Since $y_j = \text{Cost}_j(\boldsymbol{\theta})$ for all jobs j, we get:

$$\text{LP-objective} = \sum_j y_j - (1/4K) \cdot \sum_{e,t} z_{et} \geq (1/2) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta}).$$

Theorem 3 now follows from Lemma 1, Lemma 5, and Lemma 6.

Theorem 3. *If every machine follows Shortest Job First (SJF) policy and every job has unit weight, then the price of anarchy of the induced game is $O(\log^2(s_{\max}/s_{\min}))$, where $s_{\max}$ (resp. $s_{\min}$) is the maximum (resp. minimum) speed among all the machines.*

Interestingly, we get super-constant upper bound on the PoA because the speeds of the edges may keep on decreasing as we traverse farther along a path starting from the root-node. This is precisely the situation in the real-world fat-tree networks. In contrast, consider an instance where the edges adjacent to the root-node have the slowest speeds. In this instance, the proof of Theorem 2 implies that a job can get delayed by other jobs only on the first edge on its path. Thus, we can write a new time-indexed LP where we take into account the fractional completion times of the jobs only on these edges at depth one. This LP will give a constant approximation to the underlying optimization problem, as a path starting from the root contains exactly one edge of depth one. We can then execute the same analysis as outlined in this section to get constant PoA.

3 $O(d^2)$ PoA for Weighted Jobs

In this section we observe that when jobs can have arbitrarily different weights, the PoA of the game depends polynomially on the depth of the tree (see Appendix C).

Theorem 4. *There is an instance of the problem where if every edge follows HDF scheduling policy, then the game induced between the jobs has PoA $= \Omega(\sqrt{d})$.*

On the positive side, we can extend our dual-fitting framework to derive the following upper bound on the PoA (see Appendix D).

Theorem 5. *If every machine follows HDF scheduling policy, then the PoA of the game induced between the jobs is at most $8d^2$, where d is the depth of the tree.*

Ackowledgements. The authors thank Sungjin Im and Kamesh Munagala for helpful discussions.

References

1. M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.
2. S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1228–1241. SIAM, 2012.
3. A. Antoniadis, B. Moseley, N. Barcelo, M. Nugent, D. Cole, and K. Pruhs. Packet forwarding algorithms in a line network. 2014.
4. Y. Azar, K. Jain, and V. Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 323–332, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

5. N. Bansal and J. Kulkarni. Minimizing flow-time on unrelated machines. *CoRR*, abs/1401.7284, 2014.
6. S. Bhattacharya, S. Im, J. Kulkarni, and K. Munagala. Coordination mechanisms from (almost) all scheduling policies. In *5th Innovations in Theoretical Computer Science Conference*, ITCS '14, 2014.
7. V. Bilò, A. Fanelli, and L. Moscardelli. On lookahead equilibria in congestion games. In *WINE*, pages 54–67, 2013.
8. I. Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. In *SODA*, pages 815–824, 2009.
9. C. Chekuri and S. Khanna. Approximation algorithms for minimizing average weighted completion time, 2004.
10. G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. *Theor. Comput. Sci.*, 410(36):3327–3336, Aug. 2009.
11. G. Christodoulou, K. Mehlhorn, and E. Pyrga. Improving the price of anarchy for selfish routing via coordination mechanisms. In *ESA*, pages 119–130, 2011.
12. R. Cole, J. R. Correa, V. Gkatzelis, V. Mirrokni, and N. Olver. Inner product spaces for minsum coordination mechanisms. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 539–548, New York, NY, USA, 2011. ACM.
13. B. Farzad, N. Olver, and A. Vetta. A priority-based model of routing. *Chicago J. Theor. Comput. Sci.*, 2008, 2008.
14. D. G. Harris and A. Srinivasan. Constraint satisfaction, packet routing, and the lovasz local lemma. In *STOC*, pages 685–694, 2013.
15. M. Hoefer, V. S. Mirrokni, H. Röglin, and S.-H. Teng. Competitive routing over time. In *WINE*, pages 18–29, 2009.
16. S. Im and M. Ben. Scheduling over tree. *Personal communication*, 2014.
17. N. Immorlica, L. E. Li, V. S. Mirrokni, and A. S. Schulz. Coordination mechanisms for selfish scheduling. *Theor. Comput. Sci.*, 410(17):1589–1598, Apr. 2009.
18. D. Karger, C. Stein, and J. Wein. Scheduling algorithms, 1997.
19. R. Koch, E. Nasrabadi, and M. Skutella. Continuous and discrete flows over time - a general model based on measure theory. *Math. Meth. of OR*, 73(3):301–337, 2011.
20. E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.
21. E. Koutsoupias and K. Papakonstantinopoulou. Contention issues in congestion games. In *ICALP (2)*, pages 623–635, 2012.
22. F. T. Leighton, B. M. Maggs, and S. Rao. Universal packet routing algorithms (extended abstract). In *FOCS*, pages 256–269, 1988.
23. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, 34(10):892–901, 1985.
24. B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. In *WAOA*, pages 217–228, 2009.
25. T. Rothvoß. A simpler proof for $o({Congestion} + {Dilation})$ packet routing. In *IPCO*, pages 336–348, 2013.
26. T. Roughgarden. Intrinsic robustness of the price of anarchy. In *STOC*, pages 513–522, 2009.
27. T. Roughgarden and É. Tardos. How bad is selfish routing? In *FOCS*, pages 93–102, 2000.
28. W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, (3):5966, Aug. 1956.

A Justifying Assumption 1

Proof follows by first decreasing the speed on all edges by a factor of 2 and then rounding up the speeds to the nearest power of 2. By Lemma 12 (see Section D.1), if speeds

on each edge is decreased by a factor of 2, then the cost of solution increases by a factor of 2. Let x^* be a solution to LP 23, where speed on each edge $s(e)$ is scaled down by a factor of 2. Now, round up the speed $s(e)/2$ to nearest power of 2. Observe that x^* continues to be a feasible solution to this instance of the problem as the speeds on each edge only increased. This completes the proof.

B Complete Proof of Theorem 2

Consider the path $\boldsymbol{\theta}_{j^*} = (e_1, \ldots, e_l)$, where e_1 (resp. e_l) is the edge adjacent to (resp. farthest away from) the root. We decompose this path in w segments, for some natural number w, in the following manner. Consider a function $f : [1, w+1] \to [1, l+1]$ such that $1 = f(1) < f(2) < \cdots < f(w+1) = l+1$. Segment $k \in [1, w]$ corresponds to the sequence of edges $e_{f(k)}, e_{f(k)+1}, \ldots, e_{f(k+1)-1}$. The decomposition satisfies two properties.

- The speeds of the first edges of these segments form a strictly decreasing sequence. Hence, Assumption 1 implies that:

$$s(e_{f(1)}) > s(e_{f(2)}) > \cdots > s(e_{f(k)}) \tag{15}$$

- Within each segment, the first edge has the slowest speed. Thus, for each $k \in [1, w]$:

$$s(e_{f(k)}) \leq s(e) \quad \text{for all } e \in \{e_{f(k)+1}, \ldots, e_{f(k+1)-1}\} \tag{16}$$

Let J_k be the set of jobs whose paths (under the outcome $\boldsymbol{\theta}$) contain the first edge in segment k, that is, $J_k = \{j : e_{f(k)} \in \boldsymbol{\theta}_j\}$ for all $k \in [1, w]$. Note that these sets form a laminar family, i.e., $J_1 \supseteq J_2 \supseteq \cdots \supseteq J_w$. Let $t_k(j)$ be the time at which job $j \in J_k$ arrives at the first edge of segment k. Equivalently, it is the time at which the job is completely processed on the previous edge $e_{f(k)-1}$. For $k = 1$, we have $t_k(j) = 0$. We will prove the following lemma.

Lemma 7. *Consider any segment $k \in [1, w]$, and order the jobs in J_k according to their lengths, starting from the smallest job. Let $J_k = \{j_1, \ldots, j_\eta\}$, where $\eta = |J_k|$, and $p_{j_1} < \cdots < p_{j_\eta}$. The jobs satisfies two properties.*

- *(P1) We have $t_k(j_1) \leq \cdots \leq t_k(j_\eta)$.*
- *(P2) A job $j \in J_k$ can delay another job $j' \in J_k$ only on the first edge $e_{f(k)}$.*

We will prove Lemma 7 by induction. Clearly, property (P1) holds for segment 1. The induction hypothesis is stated below.

Assumption 6 *Both the properties (P1) and (P2) hold till segment $k-1$, and furthermore, the property (P1) holds for segment k.*

Now, Lemma 7 follows from the proof of Lemma 8 (see Section B.1).

Lemma 8. *If Assumption 6 holds, then the property (P2) holds for segment k, and the property (P1) holds for segment $k+1$.*

We proceed with the proof of Theorem 2. Consider the decomposition of the path $\boldsymbol{\theta}_{j^*}$ into w segments (see Equation 15 and Equation 16), and let k^* be the segment with the largest index whose first edge also belongs to the path $\boldsymbol{\theta}_{j_1^*}$. Specifically, we have

$$k^* = \max\{k \in [1, w] \,:\, e_{f(k)} \in \boldsymbol{\theta}_{j_1^*}\}.$$

From Equation 15 and Equation 16, it follows that $e_{f(k^*)} \in \boldsymbol{\theta}_{j^*} \cap \boldsymbol{\theta}_{j_1^*}$, and that $e_{f(k^*)}$ is the edge with slowest speed in $\boldsymbol{\theta}_{j^*} \cap \boldsymbol{\theta}_{j_1^*}$. By Lemma 7, the job j^* can get delayed by the job j_1^* only on the first edges of the segments $1, \ldots, k^*$. We ignore the remaining segments as the path taken by the job j_1^*, by definition, does not overlap with them. Now, under SJF policy, the job j^* can get delayed by another job j_1^* only if $p_{j_1^*} \leq p_{j^*}$, and, in this case, the delay experienced by j^* due to j_1^* on any specific edge e' is at most $p_{j_1^*}/s(e') \leq p_{j^*}/s(e')$. Thus, the total delay of the job j^* due to the job j_1^* is at most:

$$\sum_{t=0}^{k^*-1} \frac{p_{j^*}}{s(e_{f(k^*-t)})} \leq \sum_{t=0}^{k^*-1} \left(\frac{1}{2^t}\right) \times \frac{p_{j^*}}{s(e_{f(k^*)})} \leq 2 \times \frac{p_{j^*}}{s(e_{f(k^*)})} \tag{17}$$

The first inequality holds since $s(e_{f(k^*-t)}) \geq 2^t \times s(e_{f(k^*)})$, which, in turn, holds since the speeds of the edges are discretized in powers of two (see Assumption 1) and since the speeds of the first edges of the segments form a strictly decreasing sequence (see Equation 15).

Theorem 2 follows from Equation 17 and the fact that the edge $e_{f(k^*)}$ has the slowest speed among all the edges that are common to the paths $\boldsymbol{\theta}_{j^*}$ and $\boldsymbol{\theta}_{j_1^*}$.

B.1 Proof of Lemma 8

Under SJF scheduling policy, a job $j \in J_k$ can delay a job $j' \in J_k$ only if $p_j < p_{j'}$. For ease of exposition, we first consider the case where the job j has the smallest length in J_k, i.e., $j = j_1$ and $j' = j_t$ for some $t > 1$. In this case, the SJF policy ensures the job j will not get delayed by any other job. Define $\Delta_{j_1} = p_{j_1}/s(e_{f(k)})$ to be the processing time of the job j_1 on the first edge of segment k.

Lemma 9. *Consider any edge $e \neq e_{f(k)}$ in segment k that is common to the paths taken by the jobs j_1, j', where $j' \in J_k$ and $p_{j_1} < p_{j'}$. The arrival time of job j' on edge e is at least Δ_{j_1} more than the arrival time of the job j_1 on the same edge.*

Proof. We will prove this lemma by induction. We start by considering the case where $e = e_{f(k)+1}$. By Assumption 6, the job j' arrives on edge $e_{f(k)}$ on or after the job j_1. As the job j_1 has higher priority under SJF policy, the job j' cannot start getting processed on the edge $e_{f(k)}$ before the job j_1 arrives at the next edge $e_{f(k)+1}$. Furthermore, since $p_{j_1} < p_{j'}$, the job j' will require at least Δ_{j_1} time-unit of processing on edge $e_{f(k)}$. Hence, the claim holds for the edge $e = e_{f(k)+1}$.

Now, consider the case where $e \notin \{e_{f(k)} \cup e_{f(k)+1}\}$. Let e^- be the ancestor of the edge e. By induction hypothesis for Lemma 9, assume that the job j_1 arrives on the edge e^- at least Δ_{j_1} units of time ahead of the job j'. Note that the job j_1 will never get delayed by another job on edge e^-, and that $p_{j_1,e^-} \leq p_{j',e^-}$. Hence, Lemma 9 holds for the edge e as well, and this concludes the proof of the induction step.

Lemma 10. *Consider any edge $e \neq e_{f(k)}$ in segment k that is common to the paths taken by the jobs j_1, j', where $j' \in J_k$ and $p_{j_1} < p_{j'}$. The job j_1 exits the edge e before the arrival of the job j' on the same edge.*

Proof. Consider the time t when the job j_1 arrives on the edge e. Since the job j_1 has highest priority, it will exit the edge at time $t + p_{j_1}/s(e)$. By Claim 9, the job j' arrives on the edge e at time $t + \Delta_{j_1}$. Since $e_{f(k)}$ is slowest edge in the segment, we have $\Delta_{j_1} \geq p_{j_1}/s(e)$, and hence $t + \Delta_{j_1} \geq t + p_{j_1}/s(e)$. The lemma follows.

Lemma 9 and Lemma 10 imply that in segment k, the job j_1 can delay any other job $j' \in J_k \setminus \{j_1\}$ only on the first edge $e_{f(k)}$ (property (P2) for segment k), and leaves segment k before every other job in $J_k \setminus \{j_1\}$ (property (P1) for segment $k+1$). In other words, we can remove the job j_1 from our consideration after it is completely processed on the first edge of segment k, as it will not meet any other job in the subsequent edges of the same segment.

Keeping this in mind, we can simply replace j_1 by j_2 in the statements of Lemma 9 and Lemma 10. The arguments used to derive these lemmas will remain unchanged. Accordingly, we can infer that in segment k, the next job j_2 can delay any other job $j' \in J_k \setminus \{j_1, j_2\}$, only on the first edge $e_{f(k)}$. Furthermore, the job j_2 exits the segment before every other job in $J_k \setminus \{j_1, j_2\}$.

We now remove the job j_2 from our consideration and consider the job j_3 and the remaining set of jobs $J_k \setminus \{j_1, j_2, j_3\}$. Proceeding along these lines, we derive Lemma 8.

C Proof of Theorem 4

Consider the following rooted tree (T, r). From the root node, there is a path of length d leading to a leaf node l_1. Each edge on this path has a speed of 1. The root node r also has d unit length paths to leaves l_i for $i \in [2, 3, \ldots d+1]$. Each edge in these paths has a speed of $\frac{1}{d\sqrt{d}}$. There is one big job j^* with weight $\sqrt{d}$ and processing length $\sqrt{d} - \epsilon$ for some arbitrary small constant ϵ. There is another set of d jobs each having unit processing length and unit weight. Each job wants to go to any one of the leaf-nodes.

Note that the density of job j^* is slightly higher than rest of the jobs. Consider the following Nash Equilibrium. The job j^* chooses the path of length d and small jobs choose the path of length one to the leaves. It is easy to verify that no small jobs wants to deviate as any such deviation will only increase their sojourn time as j^* is a higher density job and they have to wait till it gets processed on each edge on the path. Similarly, the job j^* does not want to deviate to paths of lengths one since its sojourn time will be $\sqrt{d} \cdot \frac{1}{d\sqrt{d}} = d^2$ on them while its sojourn time on the longer path is $\sqrt{d} \cdot d = d\sqrt{d}$. It is easy to verify that, total cost of this NE is at least $d^2\sqrt{d}$.

Now consider the following solution to this instance. All the jobs are routed through the long path but jobs with legnth 1 get routed first before j^* is routed. In this solution all the small jobs have the sojourn time at most $2d$ and the sojourn time of big job j^* is $d + d\sqrt{d}d + d\sqrt{d}$. This follows as the big job has to wait d units of time on the first edge of the path to leaf l_1 and after that it never gets delayed by small jobs on the subsequent edges and it takes $\sqrt{d}$ time to get processed. Hence total weighted sojourn time of all

jobs is at most $d * 2d + \sqrt{d} \cdot (d + d\sqrt{d}) \leq 4d^2$. This shows that price of anarchy is at least $\sqrt{d}$.

Next we show that, scheduling policies such as Round Robin (RR) (or Shortest Elapsed Time First (SETF)) lead to games with PoA of $\Omega(d)$ even when jobs are unit weight and unit length and the tree is a simple path of length d. The following theorem implies that, scheduling policies like RR are not good for minimizing the weighted completion time in the case of simple networks. This shaprly contrasts with $O(1)$ approximation and PoA of these policies even in the unrelated machine scheduling context.

Theorem 7. *If scheduling policy declared by each edge of a **path** is Round Robin, then approximation ratio of the resulting solution is at least $\Omega(d)$ even when jobs are unit weight and unit length.*

Proof. Consider the following instance. The instance consists of a set of d unit length, unit weight jobs. There is a path of length d with u as the left most vertex and v as the right most vertex. All the jobs arrive at u at the time $t = 0$ and want to reach v. If the scheduling policy declared by each edge is Round Robin, then each job finishes exactly as the same time as every other job at all the edges and hence all the jobs reach vertex v at the time d^3. In the optimal solution, each edge follows Shortest Job First and hence completion time of jobs is at most $O(d^2)$. This completes the proof.

D Proof of Theorem 5

We consider an LP-relaxation for the underlying optimization problem. In a feasible integral solution of LP (18), the variable $x_{ij} \in \{0, 1\}$ is set to 1 iff the job j takes the path $i \in \mathcal{L}_j$. Similarly, the variable $x_{ijet} \in \{0, 1\}$ is set to 1 iff the job j takes the path $i \in \mathcal{L}_j$ and is being processed on the edge $e \in i$ at time-step t.

To understand the LP-objective, fix any job j and suppose that it selects path $i \in \mathcal{L}_j$ (If j does not select path i, then the corresponding x_{ijet} terms in the objective are all zero). For each $e \in i$, let C_{je} denote *j's completion time on e*: it is the time at which the edge e finishes processing the job j. Further, let C_j denote *j's sojourn-time*: it is the time at which the leaf-edge on path i finishes processing j. Finally, *j's fractional completion time on the edge e* is defined as $F_{je} = \sum_t x_{ijet} \cdot (t/p_{je})$.

It is easy to verify that $F_{je} \leq C_{je} \leq C_j$ for all $e \in i$. As the tree has depth d, there are most d edges on path i, and accordingly, we get $(1/d) \sum_{e \in i} F_{je} \leq C_j$. Multiplying both sides by w_j and summing over all the jobs, we see that the first half of the LP-objective is at most $\sum_j w_j \cdot C_j$.

Next, note that the total amount of processing done on j is a lower bound on C_j, that is, $\sum_{e \in i} \sum_t x_{ijet} \leq C_j$. As before, multiplying both sides by w_j and summing over all the jobs, we see that the second half of the LP-objective is also upper bounded by $\sum_j w_j \cdot C_j$. Thus, the overall LP-objective is at most $2 \times \sum_j w_j \cdot C_j$.

$$\text{Minimize} \quad \sum_j \sum_{i \in \mathcal{L}_j} \frac{1}{d} \cdot \sum_{e \in i} \sum_t w_j x_{ijet} \cdot \left(\frac{t}{p_{ej}} \right) + \sum_j \sum_{i \in \mathcal{L}_j} \sum_{e \in i} \sum_t w_j x_{ijet} \quad (18)$$

$$\sum_{i \in \mathcal{L}_j} x_{ij} \geq 1 \qquad \forall \text{ jobs } j \tag{19}$$

$$\sum_{t} \frac{x_{ijet}}{p_{ej}} \geq x_{ij} \qquad \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j, \text{ edges } e \in i \tag{20}$$

$$\sum_{j} \sum_{i \in \mathcal{L}_j : e \in i} x_{ijet} \leq 1 \qquad \forall \text{ edges } e, \text{ times } t \tag{21}$$

$$x_{ijet}, x_{ij} \geq 0 \qquad \forall j, i \in \mathcal{L}_j, e \in i, t \tag{22}$$

Constraint 19 states that every job has to take some path. Constraint 20 states that if a job j takes a path i, then it has to get completely processed on all the edges on this path. Finally, constraint 21 states that a machine can process at most one unit of the jobs during any unit time-step. Thus, we get the following lemma.

Lemma 11. *The optimal objective of LP (18) is at most* 2 *times the optimal objective of the underlying optimization problem.*

We now get LP (23) by changing the right hand side of constraint 21 to $1/2d$. This imposes the condition that a machine can process at most $1/2d$ units of the jobs during one time-step, and increases the objective by a factor of $2d$.

$$\text{Minimize} \quad \sum_{j} \sum_{i \in \mathcal{L}_j} \frac{1}{d} \cdot \sum_{e \in i} \sum_{t} w_j x_{ijet} \cdot \left(\frac{t}{p_{ej}} \right) + \sum_{j} \sum_{i \in \mathcal{L}_j} \sum_{e \in i} \sum_{t} w_j x_{ijet} \tag{23}$$

$$\sum_{i \in \mathcal{L}_j} x_{ij} \geq 1 \qquad \forall \text{ jobs } j \tag{24}$$

$$\sum_{t} \frac{x_{ijet}}{p_{ej}} \geq x_{ij} \qquad \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j, \text{ edges } e \in i \tag{25}$$

$$\sum_{j} \sum_{i \in \mathcal{L}_j : e \in i} x_{ijet} \leq 1/2d \qquad \forall \text{ edges } e, \text{ times } t \tag{26}$$

$$x_{ijet}, x_{ij} \geq 0 \qquad \forall j, i \in \mathcal{L}_j, e \in i, t \tag{27}$$

Lemma 12. *The optimal objective of LP (23) is at most* $4d$ *times the optimal objective of the underlying optimization problem.*

Proof. The proof of this lemma follows from *time stretching* arguments used in minimizing sum of completion time results. Consider a solution x^* to 18. Divide each unit of time t into $2d$ time slots $\{t'_1, t'_2, \ldots t'_{2d}\}$ and distribute the total processing done at t across these slots such that total volume of jobs procesed at each time slot $t'_i, i \in [2d]$ is at most $1/2d$. Now, consider the solution y^* to 18 by treating each $t'_i, i \in [2d]$ as an unit of time. Clearly, y^* is a feasible solution to 23. By mapping each time instant t in x^* to time instant $2d \cdot t$ in y^*, it is easy to see that the cost solution y^* is at most $2d$ times the cost of solution x^*. This completes the proof.

Below, we write down the dual of LP (23).

$$\text{Maximize} \qquad \sum_j y_j - \frac{1}{2d} \cdot \sum_{e,t} z_{et} \tag{28}$$

$$y_j \leq \sum_{e \in i} u_{ije} \qquad \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j \tag{29}$$

$$\frac{u_{ije}}{p_{je}} - z_{et} \leq (w_j/d) \cdot \left(\frac{t}{p_{ej}}\right) + w_j \qquad \forall \text{ jobs } j, i \in \mathcal{L}_j, e \in i, \text{ times } t \tag{30}$$

$$y_j, z_{et}, u_{ije} \geq 0 \qquad \forall j, t, i \in \mathcal{L}_j, e \in i \tag{31}$$

D.1 PoA under HDF scheduling policy via dual fitting

Throughout Section D.1, we assume that every machine (edge) follows HDF scheduling policy. We analyze the PoA of the game induced between the jobs. In Figure 3, we present an algorithm that considers any pure Nash equilibrium of this game, and based on this input, assigns values to the dual variables in LP (28).

INPUT: An outcome $\boldsymbol{\theta}$ of the game that is in a pure Nash equilibrium.

1. Set $y_j \leftarrow (1/d) \cdot \text{Cost}_j(\boldsymbol{\theta})$.
2. Define $\delta_{ije}(j')$ to be the total delay experienced by job j on edge $e \in i \in \mathcal{L}_j$ due to the job j' under the outcome $(i, \boldsymbol{\theta}_{-j})$. Set $u_{ije} \leftarrow (w_j/d) \cdot p_{je} + \sum_{j' \neq j} (w_j/d) \cdot \delta_{ije}(j')$.
3. Define $\lambda_{jet} \in \{0, w_j\}$ as follows. We have $\lambda_{jet} = w_j$ iff $e \in \boldsymbol{\theta}_j$ *and* the sojourn-time of job j under outcome $\boldsymbol{\theta}$ is at least t. Otherwise, $\lambda_{jet} = 0$. Set $z_{et} \leftarrow (1/d) \cdot \sum_j \lambda_{jet}$.

Fig. 3. An algorithm for setting the dual variables in LP (28).

Lemma 13. *If the dual variables are assigned values as in Figure 3, then for all jobs j and paths $i \in \mathcal{L}_j$, we have $(1/d) \cdot \text{Cost}_j(i, \boldsymbol{\theta}_{-j}) = \sum_{e \in i} u_{ije}$.*

Proof. Fix a job j, a path $i \in \mathcal{L}_j$, and consider the outcome $(i, \boldsymbol{\theta}_{-j})$. The amount of time the job j spends on an edge $e \in i$ is: $p_{je} + \sum_{j' \neq j} \delta_{ije}(j') = (d/w_j) \cdot u_{ije}$. Summing over all the edges on path i, job j's sojourn-time under the outcome $(i, \boldsymbol{\theta}_{-j})$ is: $C_j(i, \boldsymbol{\theta}_{-j}) = (d/w_j) \cdot \sum_{e \in i} u_{ije}$. The lemma follows if multiply both sides of this equality by (w_j/d), and recall that $w_j \cdot C_j(i, \boldsymbol{\theta}_{-j}) = \text{Cost}_j(i, \boldsymbol{\theta}_{-j})$.

Lemma 14. *If the dual variables are assigned values as in Figure 3, then the constraint 29 of LP (28) is satisfied.*

Proof. To see why constraint 29 is feasible, note that its left hand side is equal to $(1/d) \cdot \text{Cost}_j(\boldsymbol{\theta})$. On the other hand, its right hand side, by Lemma 13, is equal to $(1/d) \cdot \text{Cost}_j(i, \boldsymbol{\theta}_{-j})$. Canceling $1/d$ from both sides, the constraint reduces to $\text{Cost}_j(\boldsymbol{\theta}) \leq \text{Cost}_j(i, \boldsymbol{\theta}_{-j})$, which holds since the outcome $\boldsymbol{\theta}$ is in a pure Nash equilibrium.

We now focus on proving that the algorithm in Figure 3 satisfies the constraint 30.

Lemma 15. *Fix any job j, any path $i \in \mathcal{L}_j$, any edge $e \in i$, and any time-step t. Define $\delta_{ije}(j', t)$ be the amount of delay experienced by j due to some other job j' on edge e,* on or after time t under outcome $(i, \boldsymbol{\theta}_{-j})$. *We have:* $\delta_{ije}(j', t) \leq \lambda_{j'et} \cdot (p_{je}/w_j)$.

Proof. The key difficulty in proving the lemma is that $\delta_{ije}(j', t)$ refers to the outcome $(i, \boldsymbol{\theta}_{-j})$, and, in contrast, $\lambda_{j'et}$ refers to the outcome $\boldsymbol{\theta}$. To overcome this difficulty, we define the quantity $\lambda^*_{j'et}$ to be the exact analogue of $\lambda_{j'et}$ under the outcome $(i, \boldsymbol{\theta}_{-j})$.

$$\lambda^*_{j'et} = \begin{cases} w_{j'} & \text{if } e \in \boldsymbol{\theta}_{j'} \textit{ and} \text{ the sojourn-time of } j' \text{ under } (i, \boldsymbol{\theta}_{-j}) \text{ is at least } t; \\ 0 & \text{otherwise.} \end{cases}$$

We now consider two possible cases.

Case 1. Either $\lambda^*_{j'et} = 0$ or $w_j/p_{je} > w_{j'}/p_{j'e}$.

It is easy to check that if $\lambda^*_{j'et} = 0$, then $\delta_{ije}(j', t) = 0$. Further, HDF scheduling policy ensures that $\delta_{ije}(j', t) = 0$ if the density of j is more than that of j', i.e., if $w_j/p_{je} > w_{j'}/p_{j'e}$. Hence, Lemma 15 holds in this case.

Case 2. We have (a) $\lambda^*_{jet} = w_{j'}$ *and* (b) $w_j/p_{je} \leq w_{j'}/p_{j'e}$.

Here, the delay $\delta_{ije}(j', t)$ is clearly upper bounded by $p_{j'e}$, which, by conditions (a) and (b), is at most $\lambda^*_{j'et} \cdot (p_{je}/w_j)$. Thus, we get:

$$\delta_{ije}(j', t) \leq \lambda^*_{j'et} \cdot (p_{je}/w_j) \tag{32}$$

Since j' has higher density than j, HDF scheduling policy ensures that the processing of j' is not affected if j switches its path. Specifically, the time-steps at which j' is processed by edge e remains unchanged under the two outcomes $(i, \boldsymbol{\theta}_{-j})$ and $\boldsymbol{\theta}$. Thus, we have $\lambda_{j'et} = \lambda^*_{j'et}$, and equation 32 implies that Lemma 15 holds in this case as well.

Lemma 16. *If the dual variables are assigned values as in Figure 3, then the constraint 30 of LP (28) is satisfied.*

Proof. As in constraint 30, fix a job j, a path $i \in \mathcal{L}_j$, an edge $e \in i$, and a time-step t. Lemma 15 states that for every job $j' \neq j$, we have $\delta_{ije}(j', t) \leq \lambda_{j'et} \cdot (p_{je}/w_j)$. Summing over all jobs $j' \neq j$, we get:

$$\sum_{j' \neq j} \delta_{ije}(j', t) \leq (p_{je}/w_j) \cdot \sum_{j' \neq j} \lambda_{j'et} \tag{33}$$

Next, the quantity $\sum_{j' \neq j} (\delta_{ije}(j') - \delta_{ije}(j', t))$ is the total delay experienced by j on edge e till time t, under the outcome $(i, \boldsymbol{\theta}_{-j})$. So it is clearly upper bounded by t.

$$\sum_{j' \neq j} (\delta_{ije}(j') - \delta_{ije}(j', t)) \leq t \tag{34}$$

Adding Equation 33 and Equation 34, and noting that $\sum_{j' \neq j} \lambda_{j'et} \leq d \cdot z_{et}$, we get:

$$\sum_{j' \neq j} \delta_{ije}(j') \leq t + (p_{je}/w_j) \cdot d \cdot z_{et} \tag{35}$$

From equation 35, and the way u_{ije} is assigned a value in Figure 3, we infer that:

$$u_{ije} \leq (w_j/d) \cdot p_{je} + (w_j/d) \cdot t + p_{je} \cdot z_{et} \tag{36}$$

Rearranging the terms, we get:

$$\frac{u_{ije}}{p_{je}} - z_{et} \leq (w_j/d) + (w_j/d) \cdot (t/p_{je}) \leq (w_j/d) \cdot \left(\frac{t}{p_{je}}\right) + w_j \tag{37}$$

The last inequality holds since $d \geq 1$.

Lemma 17. *If the dual variables are assigned values as in Figure 3, then all the constraints of LP (28) are satisfied.*

Proof. Follows from Lemma 14 and Lemma 16.

Lemma 18. *If the dual variables are set as in Figure 3, then the objective of LP (28) is at least* $(1/2d) \cdot \sum_j \mathit{Cost}_j(\boldsymbol{\theta})$.

Proof. The first part of the objective – $\sum_j y_j$ – clearly equals $(1/d) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta})$. Below, we focus on the second part of the LP-objective. Fix the outcome $\boldsymbol{\theta}$ for the rest of the proof. Focus on a job j with sojourn-time $C_j = (1/w_j) \cdot \text{Cost}_j(\boldsymbol{\theta})$, and consider two possible cases.

(Case 1.) $t \leq C_j$.

In this case, Figure 3 implies that the job j contributes w_j/d to each of the z_{et}'s corresponding to the edges in the path $\boldsymbol{\theta}_j$, and it makes zero contribution to the remaining z_{et}'s. Since the path $\boldsymbol{\theta}_j$ has at most d edges, d being the depth of the tree, the total contribution of the job j to the sum $\sum_e z_{et}$ is at most w_j.

(Case 2.) $t > C_j$.

In this case, Figure 3 implies that he job j contributes zero to the sum $\sum_e z_{et}$.

Summing over all time-steps, it follows that the contribution of any single job j to the sum $\sum_{et} z_{et}$ is at most $w_j \cdot C_j$, which, by definition, is equal to $\text{Cost}_j(\boldsymbol{\theta})$. Now, summing over the contributions from all the jobs, we infer that $\sum_{et} z_{et} \leq \sum_j \text{Cost}_j(\boldsymbol{\theta})$. This, along with the observation in the preceding paragraph, proves that:

$$\begin{aligned}\sum_j y_j - (1/2d) \cdot \sum_{et} z_{et} &\geq (1/d) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta}) - (1/2d) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta}) \\ &= (1/2d) \cdot \sum_j \text{Cost}_j(\boldsymbol{\theta})\end{aligned}$$

Theorem 5 now follows from Lemma 12, Lemma 17 and Lemma 18.