Journal of Scheduling manuscript No.
(will be inserted by the editor)

Scheduling Problems over Network of Machines*

Zachary Friggstad · Arnoosh Golestanian · Kamyar Khodamoradi · Christopher Martin · Mirmahdi Rahgoshay · Mohsen Rezapour · Mohammad R. Salavatipour · Yifeng Zhang

Received: date / Accepted: date

Abstract We consider scheduling problems in which jobs need to be processed through a (shared) network of machines. The network is given in the form of a graph the edges of which represent the machines. We are also given a set of jobs, each specified by its processing time and a path in the graph. Every job needs to be processed in the order of edges specified by its path. We assume that jobs can wait between machines and preemption is not allowed; that is, once a job is started being processed on a machine, it must be completed without interruption. Every machine can only process one job at a time.

The makespan of a schedule is the earliest time by which all the jobs have finished processing. The completion time of a job in a schedule is defined as the time it finishes processing on its last machine. The total completion time refers to the sum of completion times of all the jobs. Our focus is on finding schedules with the minimum sum of completion times or minimum makespan.

In this paper, we develop several algorithms (both approximate and exact) for the problem both on general graphs and when the underlying graph of machines is a tree. Even in the very special case when the underlying network is a simple star, the problem is very interesting as it models a biprocessor scheduling with applications to data migration.

1 Introduction

Scheduling problems have been studied extensively over the past several decades. In this paper, we consider a class of scheduling problems in which there is an underlying network of machines. Before stating our problem, let us start with the classical job shop scheduling problem. In job shop, we are given a collection J of n jobs and a set M of m machines. Each job j consists of a sequence of μ_j operations $O_{1j}, O_{2j}, \ldots, O_{\mu_j j}$. Operation O_{ij} takes $p_{ij} \in \mathbb{Z}_{>0}$ time units on machine $m_{ij} \in M$. A feasible schedule specifies for each job the times its operations must be performed such that each machine processes at most one operation at any time, and an

* A preliminary version of this paper appears in the Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'2017)

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program and an NSERC Discovery Grant.

Supported by NSERC.

Department of Computing Science, University of Alberta, Canada

operation is performed only if all preceding operations are already performed. We assume all jobs are available at time zero. Let C_j be the completion time of job j in a schedule. Then the makespan of the schedule is $C_{\max} = \max_j C_j$ and the weighted sum of completion time is $\sum_j w_j C_j$ where $w_j \geq 0, j \in J$ are given weights for the jobs. Two common performance measures are to find schedules with minimum makespan or minimum (weighted) sum of completion times. We refer to the latter as min-sum or weighted min-sum objective. When p_{ij}'s are all equal to p_j (i.e. independent of the machine) then we have the *identical machine* setting. Otherwise, we have the *unrelated machine* setting.

There are many special cases of job shop scheduling studied in the literature. One view of the problem that generalizes several other problems and has drawn attention more recently is when there is a specific underlying network of machines given as a graph $G = (V, E)$ [12,4]. In this setting, we assume each edge e corresponds to a machine. Each job $j \in J$ has a specific path Q_j starting at $s_j \in V$ and ending at $t_j \in V$. The path specifies the set of machines the job has to go through in a specific order (i.e. the sequence of its operations). If the graph G is a simple path $P = v_1, v_2, \ldots, v_{m+1}$ (where $v_i v_{i+1}$ corresponds to machine m_i), each $s_j = v_1$ and $t_j = v_{m+1}$ for all jobs $j \in J$, then we get the classical flow shop problem. Also note that the standard job shop problem corresponds to the setting where G is a complete graph and the paths are Hamiltonian paths visiting every vertex is some order. Another interesting special case is when we have a general graph G, but all p_{ij}'s are 1; this problem becomes the classical packet routing problem in a network (see [15,16]). There are also works when the underlying graph G is a tree or other special graphs (see [2,14,20,21]).

1.1 Previous work

The amount of previous work on these problems is simply too large to be reviewed comprehensively here. We mention only some of the work and refer the reader to the references in them. Trivial lower bounds used in many of the previous work for makespan are the congestion and dilation lower bounds. If C is the largest congestion of any machine (the maximum over all machines i of the total running time of jobs that have an operation on i) and D is the largest dilation (longest time it would take a job to perform regardless of the presence of other jobs) then $\mathcal{L} := \max\{C, D\}$ is clearly a lower bound on the makespan. For general job shop Shmoys et al. [26] presented an algorithm with the approximation ratio of $O((\log \mathcal{L})^2 / \log \log \mathcal{L})$. When jobs can be preempted (i.e. their processing can be paused in the middle of any operations to be resumed later) one can get better results (see [3]).

Acyclic job shop is a special case of job shop where no job has two operations on the same machine. For this setting, Scheideler and Feige [7] present an algorithm to schedule with makespan $O(\mathcal{L} \log \mathcal{L} \log \log \mathcal{L})$. To complement this, for acyclic job shop with identical machines, they provide a family of instances with optimum makespan $\Omega(\mathcal{L} \log \mathcal{L} / \log \log \mathcal{L})$.

The approximation in [7] is also the best known result for the case of flow shop (which is a special case of acyclic job shop). For the slightly more general setting of flow shop where each job still has to go through the machines in the order they appear but may not need to be run on all of them (i.e. only needs to be run on a subsequence of machines), Mastrolilli and Svensson [19] prove a $\Omega(\log^{1-\epsilon} \mathcal{L})$ hardness of approximation. For the flow shop problem with identical machines (also referred to as proportionate flow shop), Shakhlevich et al. [24] present a polynomial time algorithm for the weighted min-sum objective.

As mentioned earlier, for the special case of $p_{ij} = 1$ for all i, j, the problem reduces to the packet routing problem, where each job is simply a packet that takes one unit of time to travel each edge (being a machine or a router). For this, the celebrated result of Leighton et al. [15,16]

and subsequent works show that there is a schedule of length $O(\mathcal{L})$. The most recent result by Harris and Srinivasan [11] show that there exists a schedule of makespan at most $7.26 \cdot (C + D)$ (non-constructive) and an algorithm that finds a schedule of makespan $8.84 \cdot (C + D)$ [1]. More recently, Peis et al. [20] have shown that for the case of packet routing on a directed tree, one can get a schedule of makespan at most $C + D - 1$; so this implies a simple 2-approximation. They also show that the same method can be used as a subroutine in an algorithm to get a 2-approximation for undirected trees as well. For the special case of packet routing when G is simply a path and all packets go from left-to-right, [2,13] show that the schedule in which at each time step each machine (edge) processes the job that has the shortest distance to go finds the optimum solution for the min-sum objective. Similar algorithms (namely furthest-to-go first) find the optimum solution for makespan objective [13].

For packet routing for in-trees or out-trees (directed trees in which the in-degree of each node is at most one, or out-degree is at most one, respectively) results of [17] show that the furthest-to-go strategy gives optimum solution for makespan. Based on this, [20] observe that it is easy to get a 2-approximation for makespan on undirected trees (by converting the tree into a rooted tree and splitting each schedule into two stages where in the first stage all the packets must first go up and then all the packets must go down to their destination in the 2nd stage). Similar results are claimed by Kowalski et al. [14] for makespan and min-sum objective on trees.[2]

In [18,23], the authors give a general framework for a broad class of scheduling problems (using LP rounding) that shows that any approximation algorithm with ratio ρ w.r.t. the trivial lower bound $\mathcal{L}$ for makespan can be used to obtain a $2e\rho$ approximation for the min-sum objective. As a special case, this applies to the scheduling problems on networks of identical machines. We will use this result in some of our results. It is worth pointing out that some of the ideas in [18, 23] which are also used in subsequent works have similarities to the ideas of approximation of minimum latency in vehicle routing problems (like the classical minimum latency) which use an approximation for minimum k-stroll or minimum k-spanning tree (k-MST) as a subroutine (see [5] and earlier works).

More recent works have looked at some other variants of scheduling on a network. Im and Moseley [12] look at the online scheduling problem where the network is a tree. In their model, the edges are considered routers and each leaf node corresponds to a machine. Each job must start from the root and then pass through the routers to arrive at a machine to be scheduled on. Each router and machine can process one job at a time. Machines may be unrelated, but routers are identical. They present constant factor competitive approximations using constant speed-up for makespan. Bhattacharya et al. [4] look at coordination mechanism for routing problems on a tree.

1.2 Our results

All of our results are for the identical machines setting (so each job $j \in J$ has a processing time p_j, independent of the machine), where the jobs must run on the machines non-preemptively, but they may wait in the processing queue of a machine in case it is busy at the moment.

Our first result is really just some smaller observations on our part, our more interesting results are mentioned later. However, it points out an improvement for the acyclic job shop problem with identical machines, so we think it bears mentioning.

Theorem 1 *We present the results in two parts:*

[1] These two results can be interpreted as 14.52 and 17.68-approximations, respectively.

[2] They claim a 3-approximation for makespan, and a 7-approximation for the min-sum objective, but the sketch of the proof they provide for the latter seems incorrect and there is no full proof for it.

Problem	Makespan Objective	Min-sum Objective
Trees	$\boldsymbol{O(\min\{\log n, \log m, \log p_{\max}\})}$ (cf. Theorem 1)	$\boldsymbol{O(\min\{\log n, \log m, \log p_{\max}\})}$ (cf. Theorem 1)
Trees (UPT)	2 [20]	$\boldsymbol{4e}$ (cf. Theorem 1)
Acyclic Job Shop (IM)	$\boldsymbol{O(\min\{\log n\ell, \log p_{\max}\})}$ (cf. Theorem 1)	$\boldsymbol{O(\min\{\log n\ell, \log p_{\max}\})}$ (cf. Theorem 1)
	$\Omega(\mathcal{L} \log \mathcal{L}/ \log \log \mathcal{L})$ [7]	
Junction Trees	**4** (cf. Theorem 2)	$\boldsymbol{8e}$ (cf. Theorem 2)
Junction Trees (UPT)	2 [20]	**3**
Star Network	-	**7.279** (cf. Theorem 3)
Star Network (UPT)	-	**1.796** (cf. Theorem 3)
Rooted Tree	$O(1)$ for online setting [12]	**Exact Algorithm** (cf. Theorem 4)

Table 1 The summary of results. The entries in bold font indicate the results mentioned in our theorems. In the table, IM stands for *Identical Machines*, and UPT stands for *Unit Processing Time*. As for the makespan objective for star network, the results of the junction tree are still valid in this setting as well. However, no specific approximation algorithms tailored for the makespan on stars have been reported in the literature.

1. *For both makespan and min-sum objective on trees, there are polynomial time* $O(\min\{\log n, \log m, \log p_{\max}\})$*-approximation algorithms, where* $p_{\max}$ *is the maximum processing time among all jobs. If all jobs have unit processing time, then there is a polynomial time* $4e$*-approximation for the min-sum objective.*
2. *For acyclic job shop with identical machines, under both the makespan and the min-sum objective, there are* $O(\min\{\log n\ell, \log p_{\max}\})$*-approximation algorithms where* ℓ *is the maximum number of machines in a job's sequence.*

Note $p_{\max} \leq \mathcal{L}$ so this improves over the approximation for acyclic job shop in [7] by an $O(\log \log \mathcal{L})$-factor, but only for the identical machines case. Recall that [7] show existence of family of instances of acyclic job shop with identical machines having optimum makespan $\Omega(\mathcal{L} \log \mathcal{L}/ \log \log \mathcal{L})$, so the upper bound is tight within an $O(\log \log \mathcal{L})$ factor.

We should point out that earlier works [2,13] imply a 2-approximation for minimizing the makespan for identical jobs on trees. We also consider a special case of trees, called junction-trees: in this setting, the network is a rooted tree T and for each job $j \in J$, the Q_j path for j contains the root. A special junction-tree is when T is simply a star with all the jobs starting and ending at the leaves of T.

Theorem 2 *There is a* 4*-approximation for makespan objective and an* $8e$*-approximation for the min-sum objective on junction-trees. Furthermore, if all processing times are 1, there is a different 3-approximation algorithm for the min-sum objective.*

Perhaps the strongest and most technical result of our paper is for the simplest setting of star networks. We prove the following.

Theorem 3 *For the min-sum objective on stars where all the jobs start and end on leaves, there is a randomized* 7.279*-approximation algorithm. For the special case of unit processing time, there is a randomized* 1.796*-approximation algorithm*[3]*.*

This setting is more interesting than one might initially think; it is closely related to biprocessor scheduling problems studied in, say, [10]. This connection is examined more closely at the start of Section 2.

[3] The mentioned approximation ratios for randomized algorithms are in expectation.

Another related variant of the problem on trees is when each job starts at the root and may take (any) root-to-leaf node in order to be completed. So there is not a specified path of machines that job j must run on. Instead, we have to decide the path as well as how to schedule the jobs. This is the same setting as in [12] for which the authors present online algorithms. It turns out for this setting computing a schedule with the min-sum objective can, in fact, be solved in polynomial time. We call this problem *rooted-tree routing scheduling.*

Theorem 4 *For the rooted-tree routing scheduling, there is a polynomial time algorithm to compute the optimal schedule with the min-sum objective.*

Outline of the paper: We start by studying the simplest setting (star networks) and prove Theorem 3 in Section 2. The approximation algorithms for trees and junction trees as well as the observation for acyclic job shop with identical machines (Theorems 1, 2, and 4) are presented in Section 3.

2 Approximation Algorithms for Stars

In this section, we look at the min-sum objective for scheduling on a star where jobs start/end at leaves. One problem related to the scheduling problem defined on a star network is *biprocessor scheduling* or *data migration* which can be modelled as edge sum-colouring or edge sum multi-colouring [8–10]. In the data migration problem, one has to move data stored among devices in a network from one configuration to another. The network is modelled as a graph $G = (V, E)$ where each vertex $v \in V$ represents a data storage and an edge $e = v_i v_j$ represents the need to transfer data between v_i and v_j. This transfer may take p_e time units and will keep both v_i and v_j busy for that many steps. A transfer cannot be preemptive (hence, once started must run until completed) and no node v_i can be transferring data to/from more than one other data storage at the same time. So, only data transfer over edges that form a matching can happen concurrently. The goal is to find a schedule for these transfers and minimize the makespan (the time the last transfer completes) or the min-sum objective (the average time the transfers are completed).

This is essentially biprocessor scheduling where the nodes are the processors, the tasks are represented by edges, and each task requires two specific resources (its two end-points) in order to run. When all p_e's are one, minimizing the min-sum objective is equivalent to the min-sum edge colouring of G [10], and it has been studied extensively. In the min-sum edge colouring, one has to find a proper edge colouring $\phi : E \to \mathbb{Z}^+$ that minimizes $\sum_e \phi(e)$. One can think of $\phi(e)$ as the time step in which edge e is scheduled to run on the two processors of its end-points. In the min-sum edge multi-colouring, each edge e has a requirement p_e and one has to assign p_e distinct integers (as colours) to e such that for any two adjacent edges the set of colours assigned to them are disjoint. The objective is to minimize the sum of the largest colours assigned to edges. If one further requires each set of colours to form a consecutive sequence of integers, then those p_e integers can be considered to be the time steps in which task $e = v_i v_j$ is supposed to run on the two processors v_i, v_j. The best approximation algorithm for the min-sum edge colouring is due to Halldorsson et al. [10] who present a configuration LP rounding with ratio 1.8298 and a combinatorial 1.8886-approximation. For biprocessor scheduling with arbitrary processing times p_e, Gandhi et al. [8] give a 7.682-approximation.

The problem we are considering, when restricted to networks of stars is another form of biprocessor scheduling in which each task requires being performed on two specific processors and in a specific order. More formally, suppose that the star $T = (V, E)$ with root/centre node r is the network and each job $j \in J$ starts and ends at leaf nodes s_j, t_j, respectively. For a job j, we call the transition from s_j to the root r the *first leg* of j's migration, and the transition from r to t_j the *second leg*. We first create a directed *demand* graph $H = (V_H, E_H)$ whose vertices correspond to machines (i.e. edges of T) and whose arcs correspond to jobs in J, where each arc $(s_j, t_j) \in E_H$ reflects the fact that job j needs to be processed on machines $\{s_j, r\}$ and then on $\{r, t_j\}$. So, $|V_H| = m$ and $|E_H| = n$. We will use $e_j \in E_H$ to refer to a job $j \in J$.

In this Section, we prove Theorem 3. We start first by presenting the algorithm for the general case which achieves an approximation ratio of 7.279. We then present a modified algorithm that has ratio 1.796 for when all p_j's are 1.

2.1 Approximating stars with general processing times

Our algorithm for both the general and unit processing times has the following general framework which is somewhat similar to the general framework of minimizing latency (see [5] and earlier works) to convert a makespan objective to a min-sum objective. Our algorithm works in stages where in each stage we try to find the maximum number of jobs that can be scheduled subject to a makespan bound B, which is increasing geometrically in each iteration. Lemma 1 shows how to find such a subset of jobs. We then show how even a bicriteria approximation for this makespan version of the problem can give a good approximation for the min-sum objective. This is done in Proposition 1. Most of the work is in finding a good schedule subject to the makespan bound.

Given a schedule, for a subset of jobs $\hat{J} \subseteq J$, we define the *makespan* of $\hat{J}$ as the difference in time between when the last job of $\hat{J}$ finishes processing on its last machine and when the first job of $\hat{J}$ begins processing on its first machine. We also define the *load* of a machine i with respect to the set $\hat{J}$ to be the total processing time of jobs in $\hat{J}$ incident to i in H, i.e., the congestion machine i incurs from the jobs in $\hat{J}$. Note that the notions of makespan (in our original graph T) and load (in our demand graph H) are closely related. We define (ρ, t)-proper sets of jobs, which will be used in our algorithm.

Definition 1 ($(\rho_\circ, t)$-proper set) For $\rho_\circ \geq 1$ and $t > 0$, we call a subset of jobs $\hat{J} \subseteq J$ a $(\rho_\circ, t)$-proper set if the two following conditions hold:

- $|\hat{J}|$ is at least the size of the maximum subset of J that can be scheduled with a makespan of at most t.
- For each machine i, the total load (congestion) of jobs in $\hat{J}$ that have i as their first machine (called the *in-load* of i) is at most $\rho_\circ \cdot t$ and also the load of jobs that have i as their second machine (called the *out-load* of i) is at most $\rho_\circ \cdot t$.

We, later on, show how we can build a schedule of jobs in a $(\rho_\circ, t)$-proper subset $|\hat{J}|$ with small makespan *and* small average completion time of those jobs in Proposition 1. Assuming we have an algorithm that can find $(\rho_\circ, t)$-proper sets of jobs for any given t, combined with Proposition 1 we show how we can build an algorithm for the star scheduling problem with the min-sum objective. At each iteration i, we fix a value t_i and do the following: we first find a proper set of remaining jobs with respect to t_i and then, we find a "good" scheduling of these jobs. [4] Before formally presenting the algorithm, we prove the following lemma and proposition:

Lemma 1 *There is a polynomial time algorithm that finds a $(1.5, t)$-proper set for any t.*

Proof Let OPT_t be the maximum number of jobs from J that can be scheduled with makespan at most t. First, observe that jobs/edges e in H with $p_e > \dfrac{t}{2}$ do not appear in any feasible

[4] We ideally want to find the largest set of jobs that can be scheduled at any given time t_i. However, to ensure the tractability of our algorithm, we settle for a proper set as defined instead.

scheduling with a makespan of t as each such job needs to run sequentially on two machines. We remove such jobs from consideration. Let $p_{max} = \max_j p_j$; thus $p_{max} \leq t/2$. We will find a set of jobs $\widehat{J}$ such that the in-load of each machine and the out-load of each machine is at most $t + p_{max} \leq 1.5 \cdot t$ and $|\widehat{J}| \geq OPT_t$.

To find this set, we consider the problem of picking the maximum number of jobs such that for each machine i the in-load and out-load are at most t. Note the size of this set is at least OPT_t. To find such a set, we round an LP relaxation.

Construct an undirected bipartite graph $\tilde{H} = (\tilde{V_1} \cup \tilde{V_2}, \tilde{E})$ from H: corresponding to every vertex $v \in V_H$ (i.e. for each machine), we create two copies $\tilde{v}_1$ and $\tilde{v}_2$ in $\tilde{V_1}$ and $\tilde{V_2}$, respectively; for every (directed) edge $e = (u, v) \in R_i$ (which corresponds to a job) with $p_e \leq t/2$, we put an undirected edge $\tilde{e} = (\tilde{u}_1, \tilde{v}_2)$ in $\tilde{E}$, where $\tilde{u}_1 \in \tilde{V_1}$ and $\tilde{v_2} \in \tilde{V_2}$. Let $p_{\tilde{e}}$ denote the corresponding value p_e. We work with the following LP relaxation for selecting a maximum subset of edges that imposes an in-load or out-load of at most t on the vertices:

$$\max \left\{ \sum_{e \in \tilde{E}} x_e : \sum_{e \in \delta_{\tilde{E}}(v)} p_e x_e \leq t \ \forall v \in \tilde{V_1} \cup \tilde{V_2}, \quad x \in [0,1]^{\tilde{E}} \right\} \tag{1}$$

This LP is exactly the LP relaxation for the so-called *demand matching* problem whose study was initiated in [25], where the following lemma is proved:

Lemma 2 ([25]) *Let $\boldsymbol{x}$ be an extreme point of the demand matching polytope, and let $\tilde{G}(\boldsymbol{x})$ be the graph induced by the those x_e's that have a fractional value, i.e., $0 < x_e < 1$. Then each component of $\tilde{G}(\boldsymbol{x})$ consists of a tree plus (possibly) one edge. In addition, any cycle in $\tilde{G}(\boldsymbol{x})$ has odd length.*

Now, we use a standard iterative relaxation. Similar procedures have been used in the literature on similar polytopes in the literature (e.g., see [1]). However, for the sake of completeness, we sketch the relaxation here. From Lemma 2 and the fact that the graph $\tilde{H}$ is bipartite, we have that the components of $\tilde{G}(\boldsymbol{x})$ are all trees. Now, we can round $\boldsymbol{x}$ to an integral solution $\overline{\boldsymbol{x}}$, satisfying the conditions of a $(1.5, t)$-proper set. We think of every vertex $\tilde{v} \in \tilde{V_1} \cup \tilde{V_2}$ as having an initial capacity of t for out-load (if $\tilde{v} \in \tilde{V_1}$) or in-load (if $\tilde{v} \in \tilde{V_2}$), but in the process of rounding, we allow a slight violation of that capacity. We then run an iterative process. In each iteration, for every edge e, if $x_e = 1$ or $x_e = 0$, then let $\overline{x}_e = x_e$ (fix $\overline{x}_e$), remove e from the graph, and update the capacity of the endpoints of e. As a result, every remaining edge e has a fractional x_e, and belongs to a tree. Select any edge e connecting a leaf node v to its parent in one of the trees of $\tilde{G}$, and drop the constraint for v from the LP. If the graph still has edges, solve the new LP to obtain a new optimal solution and repeat the process.It is clear that the value of the solution to the new LP is at least as good as the original LP solution since we remove a constraint. Also, since in each iteration we either fix an edge variable or remove a constraint, the number of iterations is linear. Also, as for each leaf node v (connected to the tree via edge e) whose constraint is removed, in worst case we pick e in our integer solution, the load of v in the integral solution is increased by at most $p_e \leq p_{\max}$. Therefore, this rounding algorithm terminates in linear number of iterations with a solution (set of jobs) whose size is at least as good as the solution to LP (1) and where the load of each node v is at most $t + p_{\max}$. It is straightforward to see that the edges in E corresponding to $e \in \tilde{E}$ with $\overline{x}_e = 1$ form a $(1.5, t)$- proper set. □

We should point out that the $(1.5, t)$-proper set obtained in the proof of Lemma 1 has the property that the in-load and out-load of each node is at most $t + p_{max}$. Now we describe a method that, given such a $(\rho_\circ, t)$-proper set $\hat{J}$, returns a schedule of them with a makespan of at most $2\rho_\circ \cdot t$ and furthermore, the average completion time of each job is small. While the arguments can be stated for any $\rho_\circ \geq 1$, with some hindsight, we describe the proposition for $\rho_\circ = 1.5$, and the average completion time of $\gamma \cdot t = 2.5\,t$. The use of constants $\rho_\circ$ and γ helps significantly with the readability of the calculations that will follow.

Proposition 1 *Suppose that $\hat{J}$ is a $(1.5, t)$-proper set as obtained by Lemma 1. There is a scheduling of the jobs in $\hat{J}$ with a makespan of at most $2t + 2p_{max} \leq 3t$. Furthermore, the average completion time of a job in that schedule is at most $2t + p_{max} \leq 2.5\,t$.*

Proof The algorithm for this proposition is a simple 2-stage one: in the first stage, each machine i processes (in some arbitrary order) those jobs in $\hat{J}$ that have i as their first leg, i.e. are going towards the centre of the star where this machine is the first edge they traverse. Once all the jobs in $\hat{J}$ have arrived at the centre of the star (i.e. have completed their first leg), each machine i starts processing the jobs that have i as their second machine, from smallest to largest processing time. It is straightforward to observe that each stage takes at most $t + p_{max} \leq 1.5t$ units of time to complete; so the total makespan of all jobs is at most $2t + 2p_{max} \leq 3t$.

We already showed that the schedule has makespan at most $2t + 2p_{max} \leq 3t$. We prove that the average completion time is at most $2t + p_{max}$, when $p_{max} \leq t/2$. To prove this, since the jobs have completed their first leg (arrived at the centre of the star) by time $t + p_{max}$, it is enough to show that the average completion time of each job on their second machine is at most t if we assume they were to start their second leg at time zero. So for simplicity, let us assume that all the jobs are already at the centre of the star at time zero.

Consider an arbitrary machine i and suppose that $j_1, \ldots, j_\sigma$ are the jobs in $\hat{J}$ that have i as their second machine where $p_{j_1} \leq p_{j_2} \leq \ldots \leq p_{j_\sigma}$. Based on the iterative relaxation algorithm explained earlier, we know that $\sum_{\ell=1}^{\sigma-1} p_{j_\ell} \leq t$, since the only time the rounding algorithm violates the capacity of a vertex is when it becomes a leaf of the tree and the capacity constraints for that vertex are dropped. So, w.l.o.g, we can assume that $p_{j_\sigma} = p_{\max}$. If $\sigma \leq 3$, then clearly the average completion time of each job is at most t (using the fact that each $p_j \leq t/2$). If $\sigma = 4$, then the scenario with maximum completion time is when the the jobs finish processing at times $\frac{t}{3}$, $\frac{2t}{3}$, t, and $t + p_{max}$, respectively; which implies the average completion time is at most $\frac{7t}{8}$ (note that since the second machine schedules the jobs from the smallest to the largest processing times, the worst case occurs when all the processing times are equally as large). So let us assume that $\sigma \geq 5$. It is easy to see that the completion time of job j_k (for $1 \leq k < \sigma$) is at most $(k-1)\frac{t}{\sigma-1} + p_{j_k}$. Thus, the average completion time of all the jobs on this machine will be:

$$\frac{1}{\sigma}\left(t + p_{max} + \sum_{k=1}^{\sigma-1}(k-1)\frac{t}{\sigma-1} + p_{j_k}\right) = \frac{1}{\sigma}\left((\frac{\sigma}{2}+2)t + p_{max}\right).$$

Since $\sigma \geq 5$, using the fact that $p_{max} \leq t/2$:

$$p_{max} \leq (\frac{\sigma}{2} - 2)t \implies (\frac{\sigma}{2}+2)t + p_{max} \leq \sigma t \implies \frac{1}{\sigma}\left((\frac{\sigma}{2}+2)t + p_{max}\right) \leq t,$$

as wanted. □

Now, we can formally present Algorithm 1. We have already shown how to perform Step 6, i.e. find a proper set of jobs among remaining jobs, in 1. Also, Proposition 1 gives an algorithm in Step 7 to turn the $(1.5, t_i)$-proper set found in Step 6 into a schedule for that set with a makespan of at most $3c^{i+\alpha}$ such that the average completion time of each job in that set will be $2.5c^{i+\alpha}$. The following theorem proves the approximation ratio for Algorithm 1.

```
   Data: Auxiliary graph $H$, a constant $c \in \mathbb{R}^{>0}$ to be fixed later
   Result: A scheduling of the jobs
1  $\alpha \sim U[0, 1)$
2  $i \leftarrow 1$
3  $R_1 \leftarrow E_H$;
4  while $R_i \neq \emptyset$ do
5      $t_i \leftarrow c^{i+\alpha}$
6      Find a $(1.5, t_i)$-proper subset $J_i \subseteq R_i$ (c.f. Lemma 1).
7      Schedule $J_i$ using Proposition 1, starting at the previous iteration's completion time.
8      $R_{i+1} \leftarrow R_i \setminus J_i$
9      $i \leftarrow i + 1$
10 end
```

Algorithm 1: Approximation for the min-sum scheduling on stars with identical machines.

Theorem 5 *Algorithm 1 is a* 7.279*-approximation algorithm for the min-sum objective on stars when jobs have general processing times.*

Proof In the following, whenever we refer to j'th job of a schedule, we mean that j'th job that finishes processing in that schedule. Following the notation of [5], let u_j be completion time of j'th job in our schedule and let c_j^{opt} be the completion time of j'th job in a schedule with the optimum min-sum objective (note that these jobs might not be the same). We would like to bound u_j w.r.t. c_j^{opt}. Assume that $c_j^{opt} = dc^k$ for some $d < c$ and some $k \geq 1$. Based on the value of d with respect to the random variable α in Algorithm 1, two cases arise: i) $d < c^\alpha$, or ii) $d \geq c^\alpha$. For the first case, note that since in the optimum there is a schedule of j jobs with makespan at most $c_j^{opt} = dc^k < c^{k+\alpha}$, the iteration in which the j'th job is scheduled in our algorithm is at most k. Also, note that for an iteration i, $i = 1, 2, \ldots, k-1$, the *relative* completion time of any job in iteration i with respect to i is at most $\rho c^{i+\alpha}$ where $\rho = 2\rho_\circ = 3$. By relative completion time we mean we are ignoring the offset cause by the previous iterations. The average relative completion time of each job in iteration k (using Proposition 1) is at most $\gamma c^{k+\alpha}$, where $\gamma = 2.5$. With slight abuse of notation, we bound the completion time of an arbitrary job in iteration k with its average value of $\gamma c^{k+\alpha}$ for now. This is without loss of generality, and is merely to help with the notation as we will use the expected value of the completion time soon after, which is, in fact, $\gamma c^{k+\alpha}$. Thus:

$$u_j \leq \rho \sum_{\ell=1}^{k-1} c^{\ell+\alpha} + \gamma c^{k+\alpha} \leq \frac{c^{1+\alpha}}{c-1}(\gamma c^k - \rho + (\rho - \gamma)c^{k-1}).$$

Similarly, for when $d \geq c^\alpha$, $c_j^{opt} = dc^k < c^{k+1+\alpha}$. Thus, the j'th job is scheduled no later than iteration $k+1$. Therefore:

$$u_j \leq \rho \sum_{\ell=1}^{k} c^{\ell+\alpha} + \gamma c^{k+1+\alpha} \leq \frac{c^{1+\alpha}}{c-1}(\gamma c^{k+1} - \rho + (\rho - \gamma)c^k).$$

```
   Data: Auxiliary graph $H$, a constant $c \in \mathbb{R}^{>0}$ to be fixed later
   Result: A scheduling of the jobs
 1 $\alpha \sim U[0, 1)$
 2 $i \leftarrow 1$
 3 $R_1 \leftarrow E_H$
 4 while $R_i \neq \emptyset$ do
 5     $t_i \leftarrow 2\left\lfloor \frac{c^{i+\alpha}}{2} \right\rfloor$
 6     $J_i \leftarrow$ b-Matching($t_i$)
 7     Decompose $J_i$ into $\frac{t_i}{2}$ disjoint 2-matchings $J_i^1, J_i^2, \ldots, J_i^{\frac{t_i}{2}}$ (see Lemma 3)
 8     Schedule jobs in $J_i$ according to Lemma 4
 9     $R_{i+1} \leftarrow R_i \setminus J_i$
10     $i \leftarrow i + 1$
11 end
```

Algorithm 2: Approximation for the min-sum objective on stars with identical jobs.

In the first case, $\alpha \in [\log_c d,\, 1)$ and in the second case, $\alpha \in [0,\, \log_c d)$. By taking the expectation over α over the two cases, one gets

$$\begin{aligned}
\mathbf{E}[u_j] &\leq \int_{\log_c d}^{1} \frac{c^{1+\alpha}}{c-1}(\gamma c^k - \rho + (\rho-\gamma)c^{k-1})d\alpha + \int_0^{\log_c d} \frac{c^{1+\alpha}}{c-1}(\gamma c^{k+1} - \rho + (\rho-\gamma)c^k)d\alpha \\
&= \frac{c}{c-1}\left((\gamma c^k - \rho + (\rho-\gamma)c^{k-1}) \int_{\log_c d}^{1} c^\alpha d\alpha \right. \qquad (2) \\
&\left. + \, (\gamma c^{k+1} - \rho + (\rho-\gamma)c^k) \int_0^{\log_c d} c^\alpha d\alpha \right) \\
&= \frac{c}{\ln c}\left(\gamma d c^k - \rho + (\rho-\gamma)dc^{k-1}\right) \leq \frac{c}{\ln c}\left(\gamma + \frac{\rho-\gamma}{c}\right) c_j^{opt}.
\end{aligned}$$

Setting $\rho = 3$ and $\gamma = 2.5$, and $c = 2.912$ leads to the approximation ratio of 7.279. □

2.2 Refinements for the case of unit processing times

In this section, we modify our general framework to obtain better approximation factors for the case of unit processing times. The main new ingredient of the proof is to use a different algorithm to find (ρ, t)-proper sets instead of Lemma 1. Recall that our general framework works in two steps: first, partition the jobs into disjoint blocks, and second, schedule each block separately. For unit processing time, we follow the same general framework but we use a standard b-matching algorithm for partitioning, and a more careful scheduling algorithm to deal with the jobs of each block. Algorithm 2 describes each stage more formally.

In our algorithm, the procedure b-Matching(b) finds a maximum size b-matching (a subgraph with maximum degree b) in the undirected subgraph obtained from the set of edges in R_i in polynomial time (e.g. [6]).

Lemma 3 *For even $b \geq 0$, any b-matching can be partitioned into $\frac{b}{2}$ 2-matchings.*

This is known for b-regular graphs [22]. It is straightforward to prove the same for graphs with maximum degree b as well.

Proof We will show that the b-matching is a subgraph of some b-regular graph. It has been shown that any b-regular graph (for even b) is 2 factorable [22]. This implies the claim of the lemma.

Let $F = (V_F, E_F)$ denote the graph for a b-matching and assume we have removed all the isolated vertices of F if any exists. Consider an ordering of $v_1, v_2, \ldots, v_n$ for vertices of V_F such that $1 \le deg(v_1) \le \deg(v_2) \ldots \le deg(v_n)$. We create a new graph $\hat{F}$ by making b copies $F^1, F^2, \ldots, F^b$ of F and divide its vertices into n groups of size b (each group i containing the b copies of a vertex v_i for $1 \le i \le n$). Note that there are no edges between the vertices of each group. Let $\varepsilon_i = b - deg(v_i)$ for $1 \le i \le n$. We call ε_i the *deficit* of group i. Since there are even number of vertices in each group, we can form ε_i perfect matchings on the vertices of a group i and add them to $E_{\hat{F}}$. At the end of this process, every vertex in $\hat{F}$ has a degree of b. Using the results of [22], we find the partition for $\hat{F}$. Since the original graph F is a subgraph of $\hat{F}$, the induced 2-matching on F will give us the partition claimed in the lemma. □

Next, we schedule the jobs in each block. We note that using Vizing's algorithm for edge colouring, we can schedule the jobs in J_i using $t_i + 1$ new time steps (details omitted here), however, in order to obtain a better approximation ratio we do the following. Let $\mathcal{J} = \{J_1, J_2, \ldots, J_\ell\}$ be the partitioning constructed by the algorithm, where J_i is a maximum t_i-matching. Recall that each J_i is further partitioned into 2-matchings $J_i^1, J_i^2, \ldots, J_i^{\frac{t_i}{2}}$. We call these 2-matching *slots*. Our goal is to find a scheduling of jobs in J_i (for each $i \ge 1$) with small makespan for them and at the same time small average completion time. We show how to find a schedule with makespan t_i for each J_i, $i \ge 2$ (relative to the end of the last group J_{i-1}), and with makespan $t_1 + 1$ for J_1; furthermore, for each J_i the average completion time of the jobs in J_i will be $\frac{t_i+1}{2}$. In the following lemma, we slightly abuse the definition of the makespan within each slot to refer to the number of new time units (in comparison to the previous slot) that is used to schedule its edges.

Lemma 4 *Given the partitioning $\mathcal{J}$, there exists a scheduling in which every slot J_i^t has makespan of 2, except for the very first slot J_1^1 which has a makespan of 3. The makespan of each job in J_k will be at most $1 + \sum_{\ell=1}^{k} t_k$. Furthermore, the average completion time of jobs in J_k will be at most $1 + \sum_{\ell=1}^{k-1} t_\ell + \frac{t_k+1}{2}$.*

Proof Consider an arbitrary iteration $1 \le k \le \ell$ and focus on J_k, which is partitioned further into slots $J_k^1, \ldots, J_k^{t_i/2}$. Let $\Delta_k = 1 + \sum_{\ell=1}^{k-1} t_\ell$. We want to give a schedule for jobs of J_k so that they finish in time slots $\Delta_k + 1, \ldots, \Delta_k + t_k$; this will show that the makespan of jobs in $J_1, \ldots, J_k$ will be $1 + \sum_{\ell=1}^{k} t_\ell$ for any $k \ge 1$.

Recall that each slot J_k^t accommodates a 2-matching, meaning that each connected component of J_k^t is either a path or a cycle. We consider a (directed) connected component of J_i^t which is a cycle and denote it by C. The case of a path is proved similarly. We first introduce some notation. When we associate a tuple $(\ell_u, \ell_v) \in \mathbb{Z}^+ \times \mathbb{Z}^+$ to an edge $e = (u, v)$ of a 2-matching it means that e is to be run on its first machine at time step ℓ_u and on its second machine at time step ℓ_v. Note that $\ell_u < \ell_v$. In this lemma, we develop a schedule such that $\ell_u, \ell_v \in \{1, 2, 3\}$ for the first slot of J_1 (i.e. J_1^1), and that $\ell_u, \ell_v \in \{\Delta_k + 2t - 1, \Delta_k + 2t, S\}$ for jobs in J_k^t for any other values of t and k. This implies that the jobs in J_1 are finished in times $2, \ldots, t_1 + 1$, and for each $k > 1$, jobs in J_k are finished at times $\Delta_k + 1, \ldots, \Delta_k + t_k$, where Δ_k represents the finish time of the previous block.

We simplify the problem by modifying C. Consider any maximal consecutive sequence of clockwise (counterclockwise) edges of C and contract the edges into a single "super-edge". Note that if the super-edge $e^* = (u, v)$ is associated with the tuple (ℓ_u, ℓ_v), then one can extend it to a scheduling for the entire sequence by assigning ℓ_u to the tail and ℓ_v to the head of the edges in the sequence. Every vertex of the resulting cycle has either 2 incoming or 2 outgoing edges. We call a vertex with out-degree (in-degree) 2 a *source* (*sink*) vertex. Note that the sinks are those machines that are the last machines of the 2 jobs in this slot, and the sources are the first machines of the 2 jobs. The simplified cycle has an equal number of sinks and sources, and furthermore, they must alternate (see Figure 1).

Next, we show that if C is a cycle of J_1^1, we can assign tuples $(\ell_u,\, \ell_v)$ to the edges of C where $\ell_u,\, \ell_v \in \{1,\, 2,\, 3\}$. For an edge e of C, we simply assign $(1,\, 2)$ if e is a clockwise edge and $(2,\, 3)$ otherwise. Therefore, sinks will be incident to those ends of edges with numbers 2 and 3. Similarly, the sources will touch those ends of edges associated with numbers 1 and 2. Thus, there exists no contention on the machines (no 2 jobs are scheduled at the same time on one machine), and the jobs of slot J_1^1 can all run to completion in 3 time units.

Now, assume C belongs to a slot J_k^t other than the first. We complete the proof by showing the jobs in C can be scheduled in 2 new time units with finish times in $\{\Delta_k + 2t - 1, \Delta_k + 2t\}$. Notice that in the first slot J_1^1, 3 time units were spent for scheduling at most 2 jobs on every machine. Therefore, every machine is idle for at least 1 time unit in that slot. We make use of this slack in the first slot to complete the schedule of subsequent slots in 2 additional time units instead of 3. More specifically, if we look at the union of all 2-matchings in $\{J_1, \ldots, J_{k-1}\} \cup \{J_k^1, \ldots, J_k^{t-1}\}$, the degree of each node is at most $\sum_{\ell=1}^{k-1} t_\ell + 2(t-1) = \Delta_k + 2t - 3$ whereas we have used $\Delta_k + 2t - 2$ time steps. So when consider J_k^t, at each node there must be an available time step from previous rounds (a slack) that we could schedule a job in J_k^t. For every edge in J_k^t we schedule it to run at one of $\ell_u,\, \ell_v \in \{S,\, \Delta_k + 2t - 1,\, \Delta_k + 2t\}$ time steps, where S represents the slack available at that node. That is, if the first endpoint of an edge is associated with S, we schedule that job on its first machine at an idle time unit in previous slots. Now, in a similar fashion to the case of J_1^1, we associate $(S,\, \Delta_k + 2t - 1)$ to all clockwise edges of C and $(\Delta_k + 2t - 1,\, \Delta_k + 2t)$ to counterclockwise edges. Once again, we can argue that there is no contention on the machines and that every job runs to completion in at most 2 time units. This shows that the makespan of jobs in J_k will be at most $1 + \sum_{\ell=1}^{k} t_\ell$.

In order to get small average completion time we have to slightly tweak our algorithm. For every 2-matching J_k^t where we schedule some of the tasks to be completed at time $\Delta_k + 2t - 1$ and others in time $\Delta_k + 2t$ we have the option of switching these two; i.e. when go around a cycle C (for example) and assign $(S, \Delta_k + 2t - 1)$ and $(\Delta_k + 2t - 1, \Delta_k + 2t)$ to edges in clockwise order and counterclockwise order, respectively, we can do the reverse of this choice (assign $(S, \Delta_k + 2t - 1)$ to counterclockwise and $(\Delta_k + 2t - 1, \Delta_k + 2t)$ to clockwise order). Also, when we are scheduling the slots $J_k^1, \ldots, J_k^{t_k/2}$, we can consider scheduling them in this order or the reverse order. Therefore, in each iteration i of the algorithm, to find the schedule of jobs in J_i we consider these two reverse choices (for both the ordering of two matchings as well as within each two matching) and take the better of the two. So the average completion time of the jobs in J_k is $\leq \Delta_k + \dfrac{1 + t_k}{2}$, *i.e.*, $1 + \sum_{\ell=1}^{k-1} t_\ell + \dfrac{1 + t_k}{2}$. □

The proof of the following theorem is analogous to that of Theorem 5.

Theorem 6 *Algorithm 2 is a* 1.796*-approximation algorithm for the star scheduling problem when jobs have unit processing times.*

Proof Similar to our analysis for the case of general processing times, let u_j be completion time of j'th job in our schedule and let c_j^{opt} be the completion time of j'th job in a schedule with the optimum min-sum objective. Assume $c_j^{opt} = dc^k$ for $d < c$. Similarly to the proof of Theorem 5, we consider the two cases where $d < c^\alpha$ and $d \geq c^\alpha$. In the first case, u_j is bounded from above by the amortized bound $1 + \sum_{\ell=1}^{k-1} t_\ell + \dfrac{t_k + 1}{2}$, and in the second case, by the amortized

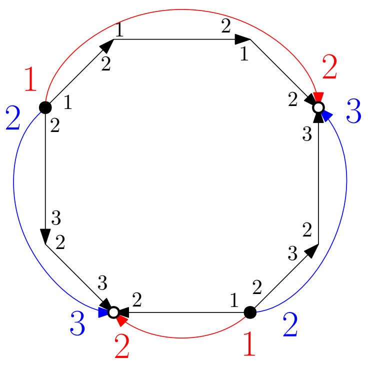

Fig. 1 Scheduling of a cycle C. The curved arrows represent the super-edges, and the solid (hollow) circles represent the source (sink) nodes. Assuming that C belongs to J_1^1, its jobs can be scheduled in 3 time units. Note that a scheduling for the super-edges can readily be extended to a scheduling for the original cycle.

bound $1 + \sum_{\ell=1}^{k} t_\ell + \dfrac{t_{k+1}+1}{2}$, where $t_\ell = 2\left\lfloor \frac{c^{\ell+\alpha}}{2} \right\rfloor$. Note that, in the first case, $1 + \sum_{\ell=1}^{k-1} t_\ell$ correspond to the sum of completion times of all the jobs in previous blocks (Δ_k), and the last term, $(t_k+1)/2$, corresponds to the amortized completion time of job j in the last block (note that the jobs in the last block have completion times of 1, 2, …, t_k). A similar arguments hold for the second case. Simplifying the bound in the first case, we get

$$\begin{aligned} u_j &\le c^\alpha + c^{1+\alpha} + \sum_{\ell=2}^{k-1} c^{\ell+\alpha} + \frac{c^{k+\alpha}+1}{2} + 1 - c^\alpha - c^{1+\alpha} + 2\left\lfloor \frac{c^{1+\alpha}}{2} \right\rfloor \\ &= \sum_{\ell=0}^{k-1} c^{\ell+\alpha} + \frac{c^{k+\alpha}}{2} + \frac{3}{2} + \beta(\alpha) = c^{k+\alpha}\left(\frac{1}{c-1} + \frac{1}{2}\right) - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta(\alpha), \end{aligned}$$

where $\beta(\alpha) = 2\left\lfloor \frac{c^{1+\alpha}}{2} \right\rfloor - c^\alpha - c^{1+\alpha}$. For the second case, we obtain the following:

$$u_j \le c^{k+1+\alpha}\left(\frac{1}{c-1} + \frac{1}{2}\right) - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta(\alpha).$$

Taking the expectation of u_j over α, we get

$$\begin{aligned} \mathbf{E}[u_j] \le& \int_{\log_c d}^{1} \left(c^{k+\alpha}\frac{c+1}{2(c-1)} - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta(\alpha)\right) d\alpha + \qquad (3) \\ & \int_{0}^{\log_c d} \left(c^{k+1+\alpha}\frac{c+1}{2(c-1)} - \frac{c^\alpha}{c-1} + \frac{3}{2} + \beta(\alpha)\right) d\alpha \\ =& \frac{c+1}{2(c-1)} c^k \int_{\log_c d}^{1} c^\alpha d\alpha + \frac{c+1}{2(c-1)} c^{k+1} \int_{0}^{\log_c d} c^\alpha d\alpha + \qquad (4) \\ & \int_0^1 \left(-\frac{c^\alpha}{c-1} + \frac{3}{2} + \beta(\alpha)\right) d\alpha \\ =& \frac{c-1}{\ln c} \cdot \frac{c+1}{2(c-1)} d c^k - \frac{1}{\ln c} + \frac{3}{2} + \int_0^1 \beta(\alpha) d\alpha. \qquad (5) \end{aligned}$$

It remains to bound $\int_0^1 \beta(\alpha)d\alpha = \int_0^1 \left(2\left\lfloor \frac{c^{1+\alpha}}{2} \right\rfloor - c^\alpha - c^{1+\alpha}\right) d\alpha$. Observe that $\lfloor c^{1+\alpha}/2 \rfloor = \kappa$ where $\kappa \in \{1, \ldots, 6\}$ is such that $1 + \alpha \in [\log_c 2\kappa, \min\{2, \log_c 2(\kappa+1)\})$ for $3 \le c < \sqrt{14}$. The range for parameter c is chosen with some foresight. Therefore,

$$\int_0^1 2\left\lfloor\frac{c^{1+\alpha}}{2}\right\rfloor d\alpha = 2\left(\int_0^{\log_c 4-1} 1d\alpha + \int_{\log_c 4-1}^{\log_c 6-1} 2d\alpha + \ldots + \int_{\log_c 12-1}^{1} 6d\alpha\right)$$
$$= 22 - 2\log_c 23040.$$

Finally,

$$\int_0^1 \beta(\alpha)d\alpha = \int_0^1 \left(2\left\lfloor\frac{c^{1+\alpha}}{2}\right\rfloor - c^\alpha - c^{1+\alpha}\right) d\alpha = 22 - 2\log_c 23040 - \frac{c-1}{\ln c} - \frac{c(c-1)}{\ln c}.$$

Substituting this value in Equation (5) and simplifying, we get

$$u_j \leq \frac{c_j^{opt}(c+1)}{2\ln c} + \frac{47}{2} - 2\log_c 23040 - \frac{c^2}{\ln c} \leq \frac{c_j^{opt}(c+1)}{2\ln c},$$

where the second inequality holds because $\frac{47}{2} - 2\log_c 23040 - \frac{c^2}{\ln c}$ is a negative term for $c > 0$. For $c = 3.59$, we obtain the claimed approximation ratio of 1.796. □

3 Scheduling on Trees and General Networks

In this section, we first focus on situations where the topology of the machines is a tree and then on the general acyclic job shop setting. We prove Theorems 1, 2, and 4.

We first recall a result from [18,23] that shows how to convert an approximation for the makespan objective that is relative to the lower bound $\max\{C, D\}$ into an approximation for the weighted min-sum objective losing only an additional constant factor. Here, C is the congestion and D is the dilation of the input. The statement below paraphrases their result.

Theorem 7 ([18,23]) *Consider an instance of job shop scheduling with jobs J having weights $w_j \geq 0, j \in J$. Suppose for any $J' \subseteq J$ we can find a schedule of J' in polynomial time having makespan $\gamma \cdot \max\{C(J'), D(J')\}$ where $C(J')$ is the maximum congestion of an edge under jobs J' and $D(J')$ is the dilation of J'. Then in polynomial time, we can find a schedule for all of J where the weighted completion time is at most $2e\gamma$ times the minimum possible weighted completion time.*

When we invoke this, we will simply have proved that for the given instance we can schedule all jobs with makespan bounded by a factor of $\max\{C, D\}$. But it should be obvious that we would get the analogous bound if we restricted to any subset of jobs because that restricted instance falls in the same family of instances we are considering (e.g. on a tree or acyclic job shop with identical machines).

3.1 Proof of Theorem 1

For showing the results of the theorem regarding trees, we invoke Theorem 2, which is proven independently of our other results (we defer the proof to Section 3.2). First, note that if all p_j's are 1, then we simply have the packet routing problem in a tree. Peis et al. [20] presented a simple algorithm in the case of directed trees that has makespan at most $C + D - 1$ (where C and D are congestion and dilation). They use their algorithm to obtain an approximation ratio of 2 for the undirected case. This, together with the result of [18,23], yields a $4e$-approximation for the min-sum objective in unit processing time.

Now, suppose that we have general processing times. We first present an algorithm with the ratio $O(\min\{\log m, \log n\})$ with respect to the two lower bounds of C, D for the makespan. Combined with Theorem 7, this yields the same approximation ratio for the min-sum objective. Finally, we focus on the acyclic job shop and present an $O(\min\{\log n\ell, \log p_{\max}\})$-approximation. This will also provide the $O(\log p_{\max})$ part of the guarantee stated in Theorem 1 for trees.

So, we now focus on trees. Let T be the underlying network. Our plan is to present an $O(\log m)$-approximation, and also an $O(\log n)$-approximation for makespan. We simply return the better of the two. For each, we decompose the problem into a logarithmic number of independent instances, each of which is the union of vertex-disjoint junction-tree instances.

To do this, we carefully select a node $v_1 \in T$ as the root (we specify how to find this vertex below) and then partition the jobs into two groups: G_1: those jobs j for which their path Q_j contains node v; and the rest are placed in $J - G_1$. Note that no job in $J - G_1$ ever needs processing on any edge incident with v_1, therefore, each such job is over a subtree of $T - v_1$. We claim that we can always pick v_1 such that the number of jobs in each of the subtrees in $T - v_1$ is at most $n/2$.

Claim Given a tree T with some subpaths $Q_1, \ldots, Q_n$ where each Q_i is a $(s_i,\, t_i)$-path for some $s_i, t_i \in V(T)$, one can always pick a vertex $v \in T$ such that the number of paths that are entirely within any subtree of $T - v$ is at most $n/2$.

Proof We argue that the claim holds by orienting the edges of the tree. For every edge $e = uv$, if more than $n/2$ of the paths Q_i are contained entirely in one subtree of $T - e$, direct e toward this subtree. Otherwise, direct e arbitrarily. After directing all edges, there is a node v that has no out-going edge. Otherwise, one could start from a leaf and follow the outgoing edges to a new vertex, each time strictly increasing the size of the subtree under the new vertex. Since there are no cycles and the outgoing edge imply that the subtree contains less that or equal to $n/2$ nodes, we should arrive at a contradiction at some point. Now it should be easy to see such e a node v exists, and has the required properties. $\square$

Trees

Note that we can find a schedule for each of the subtrees of $T - v_1$ independently and run them in parallel. Therefore, we can now solve the problem on each of those subtrees independently. For each such subtree, we pick a node as the root again; all the jobs that contain one of these roots form group G_2 and the rest of jobs belong to $J - G_1 - G_2$, and we do this recursively for each subtree. Since each time, the number of jobs left in a subtree halves, we will have at most $\log n$ iterations and hence we obtain $\sigma \le \log n$ groups $G_1, G_2, \ldots, G_\sigma$ and each group is the union of independent (i.e. vertex-disjoint) junction-tree instances. Using Theorem 2, we can obtain a 4-approximation for makespan of each group. Running these $\log n$ schedules in any arbitrary order gives an $O(\log n)$-approximation for makespan.

The algorithm for finding an $O(\log m)$-approximation is similar. We only need to pick the root v_1 (and subsequent roots) in such a way that the number of edges (i.e. machines) in each subtree left is at most half the number of edges in the original one. Such a node is commonly called a *centroid* of the tree. Therefore, we obtain $\log m$ groups this way, each of which is a collection of independent junction tree instances. Combining these we get an $O(\min\{\log n, \log m\})$-approximation for the makespan on trees and subsequently the same approximation ratio for min-sum objective function.

Acyclic Job Shop

The approximation we devise for acyclic job shop is really just a sequence of simple observations. Recall we are assuming the processing times are integers, so $p_j \ge 1$ for all jobs j. As in [7], by losing a factor of 2 in $p_{\max}, C$, and D, we assume $p_j = 2^k$ for some $k \in \mathbb{Z}_{\geq 0}$. This is achieved by scaling up all p_j to a power of 2. Observe the optimum solution value at most doubles; we could just double the start times of all operations in an optimum solution. Also, any schedule under these scaled processing times yields a schedule under the original times by using the same start times for each operation.

For each integer $0 \leq k \leq \log_2 p_{\max}$, form the group $B_k = \{j : p_j = 2^k\}$. We can view each group B_k as an instance of acyclic job shop with identical jobs, so by [16] there is a solution with makespan $O(C + D)$. More specifically, we can scale the running times of each job in B_k to be 1, which also scales the congestion and dilation by 2^{-k}. In polynomial time, we can find a schedule for these unit-length jobs with makespan $O(2^{-k} \cdot (C + D))$ [16], so under the original running times 2^k we get a solution with makespan $O(C + D)$.

Finally, we simply concatenate the resulting solutions for these $1 + \log_2 p_{\max}$ groups to get a solution for all jobs with makespan $O(\log p_{\max} \cdot (C + D))$. As this is an approximation relative to the lower bound $\max\{C, D\}$, we also get an $O(\log p_{\max})$-approximation for the min-sum objective using Theorem 7.

For the $O(\log n\ell)$-approximation, we perform the same bucketing but also form a "small job" group $B_{small} = B_0 \cup B_1 \cup \ldots \cup B_a$ where $a = (\log_2 p_{\max}) - \lceil \log_2 n\ell \rceil$. We round up *all* jobs in B_{small} to have processing time 2^a. We can solve B_{small} trivially by a greedy algorithm that simply ensures no machine is idle if it has an available job to process.

The makespan of this schedule will be at most $2^a \cdot \ell \cdot n$ because there are $\ell \cdot n$ operations in total to be performed between all jobs and at any point of time before all jobs are completed at least one machine will be busy. Note $2^a \cdot \ell \cdot n \leq p_{\max} \leq C + D$. We then solve the remaining $O(\log n\ell)$ buckets $B_{a+1}, \ldots, B_{\log_2 p_{\max}}$ as before and concatenate their schedules for a total makespan of $O(\log n\ell) \cdot (C + D))$. Again, using Theorem 7 this yields an $O(\log n\ell)$-approximation for the min-sum objective.

3.2 Proof of Theorem 2

Recall that in this setting the network of our machines forms a tree T rooted at r and the path Q_j for each job j contains r on its path.

3.2.1 General processing times

In this section, we present a 4-approximation for the makespan on junction trees which is based on the trivial lower bounds of C, D. Again, combined with the result of [18,23], this implies an $8e$-approximation for the min-sum objective function.

Let L be the value of makespan in an optimum solution. Our algorithm for makespan has two stages: in the first stage each job j moves from s_j to r; in the second stage each job j moves from r to t_j. We show how each step can be completed with makespan at most $2L$, and this yields a solution with makespan at most $4L$.

It is easier to describe the algorithm for the 2nd stage first: in this setting, all the jobs are already at the root, and the goal is to send them to their destinations (t_j's). If $u_1, \ldots, u_\sigma$ are children of r, it is enough to focus on the jobs that travel down one arbitrary edge ru_i and describe the algorithm for the subtree rooted at u_i. Suppose we sort the jobs based on their processing times from smallest to largest and start sending them (from the smallest) as soon as ru_i is free. Since each job j starts on its first edge ru_i after jobs that have smaller processing time than j, job j does not encounter delay/waiting other than at the root. Let $p_1 \leq p_2 \leq \ldots \leq p_n$ be the jobs going down ru_i. Then the maximum delay any job encounters (which happens for

```
while there is a job unfinished do
    foreach machine $e = uv$ (with $v$ being parent of $u$) do
        if $b_e(u) \neq \emptyset$ then
            process the first job in $b_e(u)$ and pass it to the next buffer;
        else if $b_e(v) \neq \emptyset$ then
            process the first job in $b_e(v)$ and pass it to the next buffer;
    end
end
```

Algorithm 3: Approximation for the min-sum objective on junction trees with unit processing times.

the last job) is $\sum_{i=1}^{n-1} p_i$ which is at most congestion C. Also, note that once j starts on the first edge, the total time it takes to complete j is exactly $|rt_j| \cdot p_j$. Noting that the largest $|rt_j| \cdot p_j$ is dilation D, all jobs are done after at most D steps, once they have started processing. Therefore, the whole makespan is at most $C + D$ which is at most $2L$.

The algorithm for sending the jobs to the root is almost the same. The best way to describe it is to consider running the same algorithm as if the jobs were supposed to start at the root and each job j is to be sent to its start point s_j. Using the same algorithm as above, all jobs can reach their designated vertex s_j in time at most $2L$. Run this schedule backwards to move all jobs j from s_j to r in time at most $2L$.

3.2.2 Special case of unit processing times

Here, we consider the case of junction trees with unit processing time and present a 3-approximation algorithm for the min-sum objective. Since we have jobs of unit processing time, we can think of the schedule a in synchronized setting were in each time step each machine starts processing one job that is available for that machine. We assume each $e = uv$ has two buffers (queues) $b_e(u)$ and $b_e(v)$ at the two ends u, v; $b_e(u)$ will buffer the jobs that arrive at u and want to cross e and $b_e(v)$ will buffer the jobs that arrive at v and want to cross u.

Our algorithm, called Algorithm 3, is very simple; it tries to keep the machines busy, giving priority to the jobs that are moving towards the root (so they are still in their first leg of their path). We show that this is a 3-approximation for the min-sum objective, which implies the 2nd part of Theorem 2.

Theorem 8 *Algorithm 3 is a 3-approximation for min-sum objective.*

We use $\delta(r)$ to denote the set of machines incident to r. For each edge e let $L(e)$ be the set of jobs whose path contains e and $l(e) = |L(e)|$. Recall that for each job j, Q_j is the unique (s_j, t_j)-path and $|Q_j|$ be the number of machines j needs to be processed on. Let OPT denote an optimum schedule and C_{OPT} the total completion time of OPT. We use C to denote the cost of our solution. In the following two lemmas, we get lower bounds for the optimum. The proof of the first lemma is immediate.

Lemma 5 $C_{\text{OPT}} \geq \sum_j |Q_j|$.

Lemma 6 $C_{\text{OPT}} \geq \sum_{e \in \delta(r)} \frac{\ell(e)(\ell(e)+1)}{4} + \frac{n}{2}$

Proof Consider an edge $e \in \delta(r)$. Clearly all jobs in $L(e)$ need to be processed on this machine. For each such job j, let $f_e(j)$ be the time e finishes processing job j in OPT (or completion time of j on e). So

$$\ell(e)(\ell(e)+1)/2 \leq \sum_{j \in L(e)} f_e(j). \tag{6}$$

For any job j, let e_1^j and e_2^j be the first and second (if it exist) machine along Q_j that are incident to the root. Obviously, the completion time of j is at least $\max\{f_{e_1^j}(j), f_{e_2^j}(j)\}$. If Q_j has two edges in $\delta(r)$, then since the finish times are all integer, we have $f_{e_2^j}(j) - f_{e_1^j}(j) \geq 1$. We say $f_{e_2^j}(j) = 0$ if Q_j has only one edge in $\delta(r)$. In either case, $\max\{f_{e_1^j}(j), f_{e_2^j}(j)\} \geq \Big(f_{e_1^j}(j) + f_{e_2^j}(j) + 1\Big)/2$. Summing this inequality over all $e \in \delta(r)$ and substituting for $f_e(j)$ from Equation (6), we conclude that $C_{\text{OPT}} \geq \frac{1}{2}(\sum_{e\in\delta(r)} \frac{\ell(e)(\ell(e)+1)}{2} + n)$, where the 1/2 in front of the summation is due to the fact that each job appears in at most two $L(e)$'s. □

Combining the above two, we obtain the following lower bound for optimum.

Corollary 1 $C_{\text{OPT}} \geq \frac{1}{3}\left(\sum_{e\in\delta(r)} \frac{\ell(e)(\ell(e)+1)}{2} + n + \sum_j |Q_j|\right)$

This corollary along with the following lemma implies Theorem 8.

Lemma 7 $C \leq \sum_{e\in\delta(r)} \frac{\ell(e)(\ell(e)-1)}{2} + \sum_j |Q_j|$.

Proof Clearly each job travels Q_j in $|Q_j|$ steps if there was no delay. It is enough to bound the total delay all the jobs incur over all edges by $\sum_{e\in\delta(r)} \ell(e)(\ell(e)-1)/2$. The main claim in the proof is to show that for any two jobs that conflict (i.e. their paths have a common edge) one unit of delay is added to the total delay of the two jobs (one of them has to wait for the other). Since any two conflicting jobs conflict on one of the edges of $\delta(r)$, the number of pairs of jobs that conflict is upper bounded by the sum above and hence that upper bounds the total delay over all jobs.

To prove the claim it is easier to think of jobs as beads; these beads move up towards the root from starting points on the first leg of their path and then continue down from the root until their destination on the second leg of their path. The beads while moving up might combine with each other to form strings: when two beads j_1 and j_2 arrive at their least common ancestor, $LCA(j_1, j_2)$ at the same time they form a string of length 2 with one going ahead of the other; say j_1 first and then j_2. This adds one unit of delay to the processing time of j_2 but whenever j_1 enters a buffer j_2 goes right behind it (i.e. it will be scheduled next after j_1). In general, for a node v with k children $u_1, \ldots, u_k$, assume that k strings of beads $S_1, \ldots, S_k$ arrive at v at (possibly) different times. Whenever the head of a string S_i arrives at v it will enter the end of the buffer at v and will save space for all the beads in its string behind itself in the buffer; any two strings that arrive at the buffer one after the other get merged into one bigger string based on the order of arrival of their heads. This will add a total of $|S_1| \times |S_2| \ldots \times |S_k|$ to the total delay of all the beads in these strings. An easy induction (which we skip here) proves this. For a job going down from the root, in worst case every job that has a conflict with it and will receive an extra delay of one for each such encounter until these two jobs pass each other over some edge. □

We conclude this section by noting that Algorithm 3 is a 2-approximation for the special case when the machines form a star. This is because by $\sum_{e\in\delta(r)} \ell(e) = 2n$ and $|Q_j| = 2$ the bounds proved in Lemmas 6 and 7 simplify to:

$$C_{\text{OPT}} \geq \sum_e \frac{\ell(e)^2}{4} + n \qquad \text{and} \qquad \mathrm{C} \leq \sum_{\mathrm{e}} \frac{\ell(\mathrm{e})^2}{2}. \tag{7}$$

Recall that for this setting our (more complicated) algorithm of Theorem 3 yields a 1.796-approximation.

3.3 Proof of Theorem 4

In this setting, each job j starts at the root and, unlike the previous settings in which a job must be processed on all machines along a given (s_j, t_j) path, it can take any path to reach any leaf node of the tree, while it has a processing time of p_j on every machine. For this case, we show that a simple greedy algorithm finds a schedule with the min-sum objective in polynomial time, hence proving Theorem 4.

Suppose $u_1, \ldots, u_d$ are the children of r. Consider an optimum solution OPT and let J_k be the set of jobs that go down a path starting at edge (machine) ru_k. The following observation is immediate:

Observation 9 *In any optimum solution, the following two hold:*

1. *The optimum solution processes the jobs in J_k in the order of their processing time from small to large.*
2. *All the jobs in J_k follow the shortest root-to-leaf path.*

Processing jobs from the smallest to the largest is known as SPT (Shortest Processing Time) rule, and it is known that on a single machine, SPT minimizes total completion time (which means it minimizes the total delay/waiting on one machine). Since using SPT there is no delay on subsequent machines for any job, it immediately implies that the optimum sends jobs down each path using SPT rule.

Let $n_k = |J_k|$ and m_k be the length of the path (number of machines from root-to-leaf) jobs in J_k travel. Suppose that the jobs in J_k from small to large are: $j_k^1, j_k^2, \ldots, j_k^{n_k}$. Since each job $j_k^a \in J_k$ will incur a delay only at the root and the delay is $p_{j_k^1} + p_{j_k^2} + \ldots + p_{j_k^{a-1}}$, and has a path of length m_k of machines to go through, the total completion time of j_k^a is $m_k p_{j_k^a} + \sum_{1 \le i \le a-1} p_{j_k^i}$. Thus, the total completion time of all the jobs in J_k is: $\sum_{1 \le i \le n_k} (m_k + n_k - i) p_{j_k^i}$, and the total completion time of all the jobs in OPT is $\sum_{1 \le k \le d} \sum_{1 \le i \le n_k} (m_k + n_k - i) p_{j_k^i}$. We use $h_k = m_k + n_k$ and call it the "load" of the branch ru_k. The following lemma follows easily.

Lemma 8 *In any optimum solution, for any two children $u_k, u_{k'}$ of r with $n_k, n'_k > 0$ we must have: $|m_k + n_k - m_{k'} - n_{k'}| \le 1$. In other words, the difference of loads of any two branches is at most 1.*

Proof By way of contradiction suppose that OPT is an optimum solution and for two children of r we have $n_k, n'_k > 0$ and $h_k \ge h_{k'} + 2$. Suppose that $J_k = j_k^1, j_k^2, \ldots, j_k^{n_k}$ and $J_{k'} = j_{k'}^1, j_{k'}^2, \ldots, j_{k'}^{n_{k'}}$ are the sequences of the jobs scheduled on branches ru_k and $ru_{k'}$, respectively. Suppose we remove job j_k^1 from branch ru_k and add it in front of the queue $J_{k'}$. The total completion time of the jobs on branch ru_k goes down by $h_k p_{j_k^1}$ and the total completion time of the jobs on branch $ru_{k'}$ goes up by $(h_{k'} + 1) p_{j_k^1}$. So the total net change in completion time is $(-h_k + h_{k'} + 1) p_{j_k^1} < 0$, which contradicts optimality of OPT. $\square$

We call a schedule in which the load of any two branches differs by at most 1 an almost balanced schedule. So the above lemma shows every optimum solution is almost balanced. We can also show the following lemma.

Lemma 9 *In any optimum solution for jobs $n, \ldots, 1$, if job 1 (the smallest job) is removed from the schedule, the remaining schedule is still an almost balanced one.*

Proof Assume J_k is the set of jobs including job 1, which are scheduled on branch ru_k. The load h_k is as big as any other branch load. To see this, suppose that job 1 is scheduled on branch ru_k with $h_k < h_{k'}$ for some other branch $ru_{k'}$ with $n_{k'} > 0$. Let i be the smallest job in $J_{k'}$ and swap 1 and i in the schedule. The net change in the total completion time will be $p_i(h_k - h_{k'}) + p_1(h_{k'} - h_k) < 0$ since $p_1 \le p_i$, which is a contradiction. $\square$

```
1  Sort the jobs in non-increasing order of their processing time, say $p_n, p_{n-1}, \ldots, p_1$;
2  Let $u_1, \ldots, u_d$ be the children of $r$; and $J_i \leftarrow \emptyset$ be the queue of jobs going down branch $ru_i$;
3  Let $m_i$ be the length of shortest root to leaf path from $ru_i$ and $n_i \leftarrow |J_i|$;
4  $j \leftarrow n$;
5  while $j \geq 1$ do
6      $k \leftarrow \text{argmin}_{1 \leq i \leq d}\{m_i + n_i\}$;
7      Schedule job $j$ in front of the queue $J_k$;
8      $n_k \leftarrow n_k + 1$;
9      $j \leftarrow j - 1$;
10 end
```

Algorithm 4: Solving the rooted-tree problem

Lemmas 8 and 9 suggest the following simple greedy algorithm which we show below finds the optimum solution. Algorithm 4 describes the greedy algorithm.

Theorem 10 *The greedy algorithm (Algorithm 4) finds an optimum solution.*

Proof We prove by backward induction on i that the greedy finds the optimum solution for the set of jobs $n, \ldots, i$ for all $n \geq i \geq 1$. The case of $i = n$ is trivial. Let $k \leq n$ be an arbitrary integer and suppose that the greedy partial schedule for jobs $n, \ldots, k+1$ is optimum for this set of jobs; call this schedule $\mathcal{S}_{k+1}$ and let $\mathcal{S}_k$ be the greedy schedule after adding job k and $\mathcal{O}_k$ be an optimum schedule for jobs $n, \ldots, k$. Let $\mathcal{O}'$ be the schedule for $n, \ldots, k+1$ obtained from $\mathcal{O}_k$ by removing job k. Since $\mathcal{S}_{k+1}$ is optimum (by hypothesis), $cost(\mathcal{S}_{k+1}) \leq cost(\mathcal{O}')$. Also, note that both $\mathcal{S}_{k+1}$ and $\mathcal{O}'$ are almost balance and have the same number of jobs. Therefore, if $h_{min}(\mathcal{O}')$ and $h_{min}(\mathcal{S}_{k+1})$ are the minimum loads in $\mathcal{O}'$ and $\mathcal{S}_{k+1}$, respectively, then $h_{min}(\mathcal{O}') = h_{min}(\mathcal{S}_{k+1})$. This implies

$$cost(\mathcal{S}_k) = cost(\mathcal{S}_{k+1}) + p_k(h_{min}(\mathcal{S}_{k+1}) + 1) \leq cost(\mathcal{O}') + p_k(h_{min}(\mathcal{O}') + 1) = cost(\mathcal{O}_k).$$

□

4 Conclusion

We have presented a number of approximations for special cases of acyclic job shop with identical machines. There are still many interesting questions one could ask.

For example, we tightened the bound between $\mathcal{L}$ and the minimum makespan for acyclic job shop with identical machines by an $O(\log\log\mathcal{L})$ factor, and now the gap is off by only an $O(\log\log\mathcal{L})$ factor. Can this be further tightened? Perhaps more interestingly, is the acyclic job shop problem with identical machines hard to approximate within any constant? It may be hard to approximate within $\Omega(\log^{1-\epsilon}\mathcal{L})$, just like flow shop with unrelated machines [19].

Are we resigned to losing logarithmic factors in trees or can we do better? Note that getting an $O(1)$-approximation for instances of acyclic flow shop with identical machines where the underlying network is a path and each job must follow a subpath is still open.

Finally, the fact that the makespan objective for acyclic job shop is super-constant hard does not necessarily mean its min-sum counterpart is also hard. By way of analogy, min-sum set cover admits a constant-factor approximation while its classic variant minimum set cover (which can be viewed as a makespan version) has a logarithmic hardness of approximation. The problem of getting either further improvements under the min-sum objective or establishing a super-constant hardness are both open.

Acknowledgement
We thank Rohit Sivakumar for preliminary discussions on this topic.

References

1. Sara Ahmadian and Zachary Friggstad. Further approximations for demand matching: Matroid constraints and minor-closed graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 55:1–55:13, 2017.
2. Antonios Antoniadis, Neal Barcelo, Daniel Cole, Kyle Fox, Benjamin Moseley, Michael Nugent, and Kirk Pruhs. Packet forwarding algorithms in a line network. In *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, pages 610–621, 2014.
3. Nikhil Bansal, Tracy Kimbrel, and Maxim Sviridenko. Job shop scheduling with unit processing times. *Math. Oper. Res.*, 31(2):381–389, 2006.
4. Sayan Bhattacharya, Janardhan Kulkarni, and Vahab S. Mirrokni. Coordination mechanisms for selfish routing over time on a tree. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 186–197, 2014.
5. Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 36–45, 2003.
6. William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization.* John Wiley & Sons, Inc., New York, NY, USA, 1998.
7. Uriel Feige and Christian Scheideler. Improved bounds for acyclic job shop scheduling. *Combinatorica*, 22(3):361–399, 2002.
8. Rajiv Gandhi, Magnús M. Halldórsson, Guy Kortsarz, and Hadas Shachnai. Improved bounds for scheduling conflicting jobs with minsum criteria. *ACM Trans. Algorithms*, 4(1):11:1–11:20, 2008.
9. Rajiv Gandhi and Julián Mestre. Combinatorial algorithms for data migration to minimize average completion time. *Algorithmica*, 54(1):54–71, 2009.
10. Magnús M. Halldórsson, Guy Kortsarz, and Maxim Sviridenko. Sum edge coloring of multigraphs via configuration LP. *ACM Trans. Algorithms*, 7(2):22:1–22:21, 2011.
11. David G. Harris and Aravind Srinivasan. Constraint satisfaction, packet routing, and the lovasz local lemma. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 685–694, 2013.
12. Sungjin Im and Benjamin Moseley. Scheduling in bandwidth constrained tree networks. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 171–180, 2015.
13. Dariusz R. Kowalski, Eyal Nussbaum, Michael Segal, and Vitaly Milyeykovski. Scheduling problems in transportation networks of line topology. *Optimization Letters*, 8(2):777–799, 2014.
14. Dariusz R. Kowalski, Zeev Nutov, and Michael Segal. Scheduling of vehicles in transportation networks. In *Communication Technologies for Vehicles - 4th International Workshop, Nets4Cars/Nets4Trains 2012, Vilnius, Lithuania, April 25-27, 2012. Proceedings*, pages 124–136, 2012.
15. Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.
16. Frank Thomson Leighton, Bruce M. Maggs, and Andréa W. Richa. Fast algorithms for finding o(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.
17. Joseph Y.-T. Leung, Tommy W. Tam, and Gilbert H. Young. On-line routing of real-time messages. *J. Parallel Distrib. Comput.*, 34(2):211–217, 1996.
18. Wenhua Li, Maurice Queyranne, Maxim Sviridenko, and Jinjiang Yuan. Approximation algorithms for shop scheduling problems with minsum objective: A correction. *J. Scheduling*, 9(6):569–570, 2006.
19. Monaldo Mastrolilli and Ola Svensson. Hardness of approximating flow and job shop scheduling problems. *J. ACM*, 58(5):20:1–20:32, 2011.
20. Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*, pages 217–228, 2009.
21. Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing on the grid. In *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, pages 120–130, 2010.

22. Julius Petersen. Die theorie der regul aren graphs. *Acta Math.*, 15:193–220, 1891.
23. Maurice Queyranne and Maxim Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5(4):287–305, 2002.
24. Natalia Shakhlevich, Han Hoogeveen, and Michael Pinedo. Minimizing total weighted completion time in a proportionate flow shop. *Journal of Scheduling*, 1(3):157–168, 1998.
25. F. Bruce Shepherd and Adrian Vetta. The demand matching problem. In *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, pages 457–474, 2002.
26. David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, 1994.