Nearly-Linear Time LP Solvers and Rounding Algorithms for Scheduling Problems

Shi Li
State Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract

We study nearly-linear time approximation algorithms for non-preemptive scheduling problems in two settings: the unrelated machine setting, and the identical machine with job precedence constraints setting, under the well-studied objectives such as makespan and weighted completion time. For many problems, we develop nearly-linear time approximation algorithms with approximation ratios matching the current best ones achieved in polynomial time.

Our main technique is linear programming relaxation. For the unrelated machine setting, we formulate mixed packing and covering LP relaxations of nearly-linear size, and solve them approximately using the nearly-linear time solver of Young. For the makespan objective, we develop a rounding algorithm with $(2+\epsilon)$-approximation ratio. For the weighted completion time objective, we prove the LP is as strong as the rectangle LP used by Im and Li, leading to a nearly-linear time $(1.45+\epsilon)$-approximation for the problem.

For problems in the identical machine with precedence constraints setting, the precedence constraints can not be formulated as packing or covering constraints. To achieve the nearly-linear running time, we define a polytope for the constraints, and leverage the multiplicative weight update (MWU) method with an oracle which always returns solutions in the polytope.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Nearly-Linear Time, Sheduling, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.86

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2111.04897`

Funding *Shi Li*: Part of the work was supported by NSF-grant CCF-1844890.

1 Introduction

Scheduling theory is an important sub-area of combinatorial optimization, operations research and approximation algorithms. Over the past few decades, advanced techniques have been developed to design approximation algorithms for numerous scheduling problems, among which mathematical relaxation is a prominent one. The algorithms based on the technique follow a two-step framework: solve some linear/convex/semi-definite programming relaxation for the problem to obtain a fractional schedule, and round it into an integral one. The main focus of the algorithm design in the literature has been the best approximation ratios that can be achieved in polynomial time. Many of the LPs used have size much larger than that of the input, and a general convex/semi-definite program requires a large polynomial time to solve, making these algorithms impractical.

To overcome the running time issue, we design approximate LP-based scheduling algorithms that run in *nearly-linear* time. We focus on two well-studied non-preemptive scheduling settings:

1. **Unrelated machine setting.** We are given a set J of n jobs, a set M of m machines, a bipartite graph $G=(M,J,E)$ between M and J, and a processing time $p_{ij}\in\mathbb{Z}_{>0}$ for every $ij\in E$, indicating the time it takes to process job j on machine i. If $ij\notin E$, then

© Shi Li;
licensed under Creative Commons License CC-BY 4.0

the job j can not be processed on machine i. The output of a problem in this setting is an assignment $\sigma \in M^J$ of jobs to machines so that $\sigma_j j \in E$ for every $j \in J$. This indicates that we process the job j on machine σ_j.

2. **Identical machine with job precedence constraints setting.** In this setting, we are given a set J of n jobs, each job $j \in J$ with a processing time $p_j \in \mathbb{Z}_{\geq 0}$, and the number $m \geq 1$ of identical machines. There are precedence constraints of the form $j \prec j'$, indicating that the job j' can only start after job j completes. The output of a problem in the setting is a completion time vector $(C_j)_{j \in J} \in \mathbb{Z}_{\geq 0}^J$, meaning that a job $j \in J$ is processed during the time interval $(C_j - p_j, C_j]$. We need $C_j \geq p_j$ for every $j \in J$, $C_j \leq C_{j'} - p_{j'}$ for every $j \prec j'$, and every integer $t \geq 1$ is contained in $(C_j - p_j, C_j]$ for at most m jobs $j \in J$.[1]

The main objective function we focus on is *weighted completion time*: We are additionally given a weight $w_j \in \mathbb{Z}_{>0}$ for every job $j \in J$, and the goal of the problem is to minimize $\sum_{j \in J} w_j C_j$, where C_j is the completion time of j on its assigned machine. For the second setting, this is explicitly given by the output. For the first setting, given the assignment $\sigma \in M^J$ of jobs to machines, it is well-known that the Smith's rule[2] gives the optimum order on each machine i. For the first setting, we also consider the objective of minimizing the *makespan*, which is defined as $\max_i \sum_{j \in \sigma^{-1}(i)} p_{ij}$, i.e., the maximum load over all machines.

It is convenient for us to use the classic three-field notation $\alpha|\beta|\gamma$ in [19] to denote scheduling problems studied in this paper.[3] The makespan and weighted completion time minimization problems in the unrelated machine setting are denoted as $R||C_{\max}$ and $R||\sum_j w_j C_j$ respectively. The problem to minimize weighted completion time in the identical machine with job precedence constraint setting is denoted as $P|\text{prec}|\sum_j w_j C_j$. We will also consider special cases of the problem, and give their notations when we discuss them.

There is a rich literature on designing approximation algorithms for these problems. For the unrelated makespan minimization problem, i.e., $R||C_{\max}$, the classic result of Lenstra, Shmoys and Tardos [32] gives a 2-approximation, which remains the state-of-the-art result. The problem is NP-hard to approximate within a factor of better than 1.5. Plotkin, Shmoys and Tardos [39] studied fast approximation algorithms for the problem, as an application of their packing and covering LP solver. They developed a randomized $(2+\epsilon)$-approximation algorithm in time $\tilde{O}_\epsilon(mn)$.[4] So their algorithm is nearly-linear if $|E| = \Theta(mn)$. Much work on the problem has focused on a special setting called the restricted assignment setting [49, 24, 25], where there is an intrinsic size $p_j \in \mathbb{Z}_{>0}$ for every $j \in J$, and for every $ij \in E$ we have $p_{ij} = p_j$.

For the unrelated machine weighted completion time problem, i.e., $R||\sum_j w_j C_j$, many independent rounding algorithms achieve an approximation ratio of 1.5 [42, 47, 43, 35]. Bansal, Svensson and Srinivasan [5] showed that the barrier of 1.5 is inherent for this type of algorithms. To overcome the barrier, they developed a novel dependent rounding scheme

[1] It is a folklore that if the last property is satisfied, we can assign $\{(C_j - p_j], j \in [J]\}$ to m machines so that the intervals assigned to each machine are disjoint.

[2] By this rule, we schedule jobs j assigned to a machine i using non-decreasing order of p_{ij}/w_j.

[3] In the notation, α indicates the machine model, β gives the set of additional constraints, and γ is the objective. $\alpha = R$ and $\alpha = P$ denote the unrelated and identical machine settings respectively, and prec $\in \beta$ indicates that jobs have precedence constraints. $\gamma = C_{\max}$ and $\gamma = \sum_j w_j C_j$ denote the makespan and weighted completion time objectives respectively.

[4] In this paper, we use $\tilde{O}_\epsilon(\cdot)$ to hide a factor that is poly-logarithmic in the input size of the instance being considered, which will be clear from the context, and polynomial in $1/\epsilon$, where ϵ is a precision parameter. An algorithm is nearly-linear if its running time is $\tilde{O}_\epsilon(\text{input size})$.

and a lifted SDP relaxation for the problem, leading to a $(1.5 - 1/2160000)$-approximation algorithm. The ratio has been improved to $1.5 - 1/6000$ by Li [35], to 1.488 by Im and Shadloo [23] and to the current best ratio of 1.45 by Im and Li [22]. The three subsequent works are based on the rectangle LP relaxation for the problem.

There is a vast literature on the problem of minimizing weighted completion time in the identical machine with job precedence constraints setting, i.e., the problem $P|\text{prec}|\sum_j w_j C_j$. A special case of the problem where there is only one machine (i.e., $m = 1$), denoted as $1|\text{prec}|\sum_j w_j C_j$, is already non-trivial. Hall et al. [20] developed a 2-approximation for the problem, which is the best possible under some stronger version of the unique game conjecture introduced by Bansal and Khot [4]. Another special case that is considered moderately in the literature is when all jobs have unit-size, denoted as $P|\text{prec}, p_j = 1|\sum_j w_j C_j$. Munier, Queyranne and Schulz [37] gave approximation ratios of 3 and 4 for the special case and the general problem $P|\text{prec}|\sum_j w_j C_j$ respectively. The ratios were improved to $1 + \sqrt{2}$ and $2 + 2\ln 2$ by Li [35]. Most algorithms [20, 37, 41, 35] for $P|\text{prec}|\sum_j w_j C_j$ and the two special cases use the following framework: Solve some linear/convex program to obtain an order of the jobs respecting the precedence constraints. For every job in this order, schedule it as early as possible, without violating the precedence and m-machine constraints.

Most of the results we discussed focused on optimizing the approximation ratios with polynomial time algorithms. Albeit being polynomial, the running times in these results are often very large. For LP-based algorithms, this may be caused by two factors. First, the size of an LP might already be large w.r.t the input size. Consider a typical time-indexed LP relaxation in the unrelated machine setting, one need a variable for every triple ijs with $ij \in E$ and s being the starting time. Assuming the number of possible starting times is linear in n, the number of variables in the LP is already $\Theta(n|E|)$; the size of the LP can only be bigger. Second, these algorithms often use a general LP solver, which has a large running time w.r.t the size of the LP. There is a vast literature in recent years on designing exact and approximate general LP solvers. Here we could only include a few representative results. To solve a linear program with $\bar{n}$ variables, $\bar{m}$ constraints and $\bar{N}$ non-zero coefficients up to a precision of ϵ, Lee and Sidford [29] developed an algorithm with running time $\tilde{O}\big((\bar{N} + \bar{m}^2)\sqrt{\bar{m}}\log\frac{1}{\epsilon}\big)$. Lee, Song and Zhang [30] gave an algorithm with running time $\tilde{O}(\bar{n}^\omega \log\frac{1}{\epsilon})$,[5] where $\omega \approx 2.373$ is the current best exponent for matrix multiplication. Brand, Lee, Sidford and Song [8] provided a $\tilde{O}(\bar{m}\bar{n} + \bar{n}^3)$ time randomized algorithm that solves the LP exactly with high probability; the running time is nearly linear if the constraint matrix is dense and tall. However, to solve general linear programs, these running times are at least quadratic, even if the LP has a linear size. Convex or semi-definite programming based algorithms need to solve the CP/SDP using the interior point or ellipsoid methods, which are often time-consuming.

1.1 Our Results

To overcome the above issue, we design approximation algorithms for scheduling problems, that run in *nearly-linear* time, i.e., in time $\tilde{O}_\epsilon(\text{input size})$. So, up to a $\text{poly}(\log n, 1/\epsilon)$-factor, our running times are the best possible. Some of the algorithms we developed have been studied empirically [2]. In the unrelated machine setting, $G = (M, J, E)$ denotes the bipartite graph between M and J, and a nearly-linear time is of order $\tilde{O}_\epsilon(|E|)$. For the identical machine with precedence constraints setting, we use κ to denote the number of precedence

[5] The result requires that the LP does not have redundant constraints.

constraints. A nearly-linear time algorithm runs in time $\tilde{O}_\epsilon(n+\kappa)$. Unlike the polynomial running time scenario, we can not assume $\prec$ is transitive, as it may dramatically increase the number of precedence constraints to quadratic. Moreover, the best known algorithm computing the transitive closure of the precedence constraints takes $O(n\kappa)$ time [40].

For many problems, including $R||C_{\max}, R||\sum_j w_jC_j, 1|\text{prec}|\sum_j w_jC_j$ and $P|\text{prec}, p_j = 1|\sum_j w_jC_j$, our nearly-linear time algorithms achieve the correspondent best known polynomial-time approximation ratios, due to Lenstra, Shmoys and Tardos [32], Im and Li [22], Hall et al. [20], and Li [35] respectively.

▶ **Theorem 1.1.** *For any $\epsilon > 0$, there is a $\tilde{O}_\epsilon(|E|)$-time $(2+\epsilon)$-approximation algorithm for $R||C_{\max}$, i.e., the makespan minimization problem on unrelated machines.*

For the problem $R||\sum_j w_jC_j$, we believe that showing that the rectangle LP can be approximated in nearly-linear time is interesting on its own. So we give two theorems for the problem. Refer to LP(6) for the formal description of the rectangle LP for the problem.

▶ **Theorem 1.2.** *Consider an instance of $R||\sum_j w_jC_j$ and the rectangle LP (6) for the instance. Let $\epsilon > 0$ and $\mathsf{lp}_{(6)}$ be the value of the LP. Then in $\tilde{O}_\epsilon(|E|)$ time, we can construct a solution $\mathbf{z}$ to the LP such that:*

- *$\mathbf{z}$ satisfies all the constraints in the LP, except that the constraint at most one job is processed on any machine at any time may be violated by a factor of $1+\epsilon$. (Formally, Constraint (8) is only satisfied with the right-side replaced by $1+\epsilon$.)*
- *The value of $\mathbf{z}$ to the LP is at most $(1+\epsilon)\mathsf{lp}_{(6)}$.*

In the theorem, our $\mathbf{z}$ will be represented by the list of non-zero coordinates and their values. Then, we show that the rounding algorithm of Im and Li [22] can indeed run in time nearly-linear on the support size of the LP solution. This gives the following theorem.

▶ **Theorem 1.3.** *For any $\epsilon > 0$, there is a $\tilde{O}_\epsilon(|E|)$-time $(1.45+\epsilon)$-approximation algorithm for $R||\sum_j w_jC_j$, i.e., the weighted completion time minimization problem on unrelated machines.*

The following two theorems are for $1|\text{prec}|\sum_j w_jC_j$ and $P|\text{prec}, p_j = 1|\sum_j w_jC_j$.

▶ **Theorem 1.4.** *For any $\epsilon > 0$, there is a $\tilde{O}_\epsilon((n+\kappa)\log p_{\max})$-time $(2+\epsilon)$-approximation algorithm for $1|\text{prec}|\sum_j w_jC_j$, i.e., the weighted completion time problem on a single machine with precedence constraints, where $p_{\max} := \max_{j\in J} p_j$ is the maximum job size.*

So the algorithm runs in nearly-linear time only when $p_{\max}$ is polynomially bounded.

▶ **Theorem 1.5.** *For any $\epsilon > 0$, there is a $\tilde{O}_\epsilon(n+\kappa)$-time $(1+\sqrt{2}+\epsilon)$-approximation algorithm for $P|\text{prec}, p_j = 1|\sum_j w_jC_j$, i.e., the weighted completion time problem on identical machines with unit-size jobs and precedence constraints.*

Along the way of algorithm design for the identical machine with precedence constraints setting, we developed a nearly-linear time $(1+\epsilon)$-approximation algorithm for the single commodity network flow problem in directed acyclic graphs, with bounded supplies and demands on sources and sinks, but infinite capacities on edges.

Recently there has been a lot of progress on solving maximum flow problem on undirected and directed graphs. For undirected graphs, the problem can be approximated within a factor of $1+\epsilon$ in nearly-linear time [26, 38, 45], and solved exactly with a slightly weaker running time of $m^{1+o(1)}$ (this is called *almost-linear* time) [7]. It was open whether an almost-linear running time can be achieved for solving maximum flow on directed graphs.[6] This was resolved in the affirmative by a recent breakthrough due to Chen et al. [14]: They developed an algorithm that computes exact maximum flows on directed graphs with polynomially bounded integral capacities in $m^{1+o(1)}$ time. Thus, we could use the result as a black-box for our problem, if we allow the running time to be almost-linear. Nevertheless as our theme is to design *nearly-linear* time algorithms, we include in the full version of the paper our approximate maximum-flow algorithm for the special case with this running time. To the best of our knowledge, this was not known before.

For the general precedence-constrained scheduling problem $P|\text{prec}|\sum_j w_jC_j$ (on multiple machines with variant job lengths), we achieve an $O(1)$-approximation algorithm in nearly-linear time. However, the approximation ratio of the algorithm is $6+\epsilon$, which is worse than the best polynomial-time ratio of $2+2\ln 2$ due to Li [35].

▶ **Theorem 1.6.** *For any $\epsilon > 0$, there is a $\tilde{O}_\epsilon((n+\kappa)\log p_{\max})$-time $(6+\epsilon)$-approximation algorithm for $P|\text{prec}|\sum_j w_jC_j$, i.e., the weighted completion time minimization problem on identical machines with precedence constraints, where $p_{\max} := \max_{j\in J} p_j$ is the maximum job size.*

1.2 Our Techniques

All of our algorithms are based on linear programming: We design an LP relaxation of nearly-linear size, solve it in nearly-linear time to obtain a $(1+\epsilon)$-approximate solution, and round the solution into an integral schedule in nearly-linear time.

For $R||C_{\max}$, the natural LP relaxation has $O(|E|)$ size, and the mixed packing and covering form. Thus it can be solved within a factor of $1+\epsilon$ by the algorithm of Young [51] in $\tilde{O}_\epsilon(|E|)$ time. In particular, the algorithm outputs a $(1+\epsilon)$-approximate solution that violates the constraints by a factor of $1\pm\epsilon$, in $O\left(\frac{\bar{N}\log\bar{m}}{\epsilon^2}\right) = \tilde{O}_\epsilon(\bar{N})$ time, where $\bar{m}$ and $\bar{N}$ are the number of constraints and non-zero coefficients in the LP respectively. To round the fractional solution, we apply the grouping technique of [46] for the so called generalized assignment problem, but with a $(1+\epsilon)$-slack. This gives us a bipartite graph $H = (V, J, E_H)$ satisfying $|N_H(J')| \geq (1+\epsilon)|J'|$ for every $J' \subseteq J$, where $N_H(J')$ is the set of neighbors of J' in H. This allows us to find a matching in H that covers J in nearly-linear time, which leads to a $(2+\epsilon)$-approximate solution, matching the current best approximation of 2 in [32]. We remark that the $\tilde{O}_\epsilon(mn)$-running time of [39] comes from both solving the LP, and rounding the LP solution. So even with the nearly-linear time mixed covering and packing LP solver, the algorithm of [39] still requires $\tilde{O}_\epsilon(mn)$ time.

For the problem $R||\sum_j w_jC_j$, we give a nearly-linear size mixed packing and covering LP that (up to a factor of $1+O(\epsilon)$) is equivalent to the rectangle LP used by Li [35], Im and Shadloo [23], Im and Li [22]. In the rectangle LP, there is a variable x_{ijs} indicating if a job j is scheduled on the machine i and has starting time s, and constraints that at most one job is processed at any time on any machine. To reduce the size of the LP to $\tilde{O}_\epsilon(|E|)$, we partition the time horizon into *windows*, with lengths geometrically increasing by a factor of $1+\epsilon$. We distinguish between two types of scheduling intervals: If a job is scheduled within a window on some machine i (we call this an inside-window interval), then we do not need to capture the precise location of the scheduling interval. On the other hand, if the job

[6] By repeatedly solving maximum flow instances on residual graphs, one can convert an approximate maximum flow algorithm on directed graphs to an exact algorithm, without much loss on the running time. So for directed graphs, allowing $(1+\epsilon)$-approximation does not give much advantage.

starts and ends at two different windows (we call the interval an cross-window interval), we will approximately capture its starting and ending times. To do so, we divide each window into $1/\epsilon$ sub-windows, and let the LP variables capture the two sub-windows containing the starting and completion times. In the LP, we require all the cross-window intervals incur a congestion of 1: any point t is covered by at most 1 fraction of cross-window intervals. Then we require the total volume of jobs processed inside each window is at most its length. We show that up to a factor of $1 + O(\epsilon)$, a solution to the LP can be converted to one for the rectangle LP with no large cost. Roughly speaking, the width of window is small compared to its position and so we do not need to know the precise location of an inside-window-interval. For a cross-window-interval, we may incur an error on its length that is about ϵ times the total length of its starting window and ending window. As a sub-window has a small length, and a cross-window-interval covers some window-boundary, the total error incurred will also be small.

We proceed to our techniques for the weighted completion time problems in the identical machine with precedence constraints setting, i.e., the problem $P|\text{prec}|\sum_j w_jC_j$ and its special cases. Due to the precedence constraints, the LP relaxations do not have the mixed packing and covering form anymore. Nevertheless, the multiplicative weight update (MWU) framework can still be applied. We enclose the precedence constraints in a polytope $\mathcal{Q}$. In each iteration of the MWU framework, we guarantee that all these constraints are satisfied, i.e., the vector we obtain is in $\mathcal{Q}$. Other than the precedence constraints, we have $\tilde{O}_\epsilon(\log p_{\max})$ packing inequalities correspondent the m-machine constraint. This is due to that we can round completion times to integer powers of $1 + \epsilon$.

The number of iterations the MWU framework takes is $\tilde{O}_\epsilon(\bar{m})$, where $\bar{m}$ is the number of packing constraints in the LP, without counting the constraints for $\mathcal{Q}$. Fortunately we have $\bar{m} = \tilde{O}_\epsilon(\log p_{\max})$. To obtain the claimed $\tilde{O}_\epsilon((n + \kappa)\log p_{\max})$ time, we need to run each iteration of MWU in nearly-linear time. The bottleneck comes from finding a vector in $\mathcal{Q}$ satisfying one aggregated packing constraint, that maximizes a linear objective with non-negative coefficients.

A key technical contribution of our paper is an oracle for the problem. For an appropriately defined directed acyclic graph $G = (V, E)$, the polytope $\mathcal{Q}$ can be formulated as $\{\mathbf{y} \in [0, 1]^V : y_v \leq y_u, \forall vu \in E\}$. For two given row vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^V_{\geq 0}$, the aggregated LP in each iteration of MWU is: $\max \mathbf{ay}$ subject to $\mathbf{y} \in \mathcal{Q}$ and $\mathbf{by} \leq 1$. Using LP duality, the problem is reduced to the special single commodity maximum flow problem we introduced: We have bounded supplies and demands on sources and sinks, but infinite capacities on edges. When allowing a $(1 + \epsilon)$-approximation for the scheduling problem, we need to find a flow whose value is at least the maximum value for the instance with sink capacities scaled by $\frac{1}{1+\epsilon}$. This is done by our nearly-linear time maximum-flow algorithm for the special case.

1.3 Other Related Work

The makespan minimization problem in the identical machine setting with precedence constraints, i.e., the problem $P|\text{prec}|C_{\max}$, is another classic problem in scheduling theory. The seminal work of Graham [18] gives a simple greedy algorithm that achieves a 2-approximation. On the negative side, Lenstra and Rinnooy Kan [31] proved a $(4/3 - \epsilon)$-hardness for the problem. Under the stronger version of the Unique Game Conjecture (UGC) introduced by Bansal and Khot [4], Svensson [48] showed that the problem is hard to approximate within a factor of $2 - \epsilon$ for any $\epsilon > 0$. Much work has focused on the special case where $m = O(1)$ and all jobs have size 1 [33, 17, 34], for which obtaining a PTAS is a long-standing open problem.

The multiplicative weight update (MWU) method for solving linear programs has played an important role in a wide range of applications. Some of its foundational work can be found in a beautiful survey by Arora, Hazan and Kale [3]. There has been a vast literature on solving packing, covering, and mixed packing and covering LPs approximately to a factor of $1+\epsilon$ using iterative methods [44, 39, 36, 50, 16, 28, 27, 51, 1, 13]. In particular, to solve a mixed packing and covering LP with $\bar{n}$ variables, $\bar{m}$ constraints and $\bar{N}$ non-zero coefficients, the algorithm of Young [51] returns $(1+\epsilon)$-approximation deterministically in $O\left(\frac{\bar{N}\ln \bar{m}}{\epsilon^2}\right)$ time. The dependence on ϵ has been improved slightly by Chekuri and Quanrud [13], who gave a randomized algorithm with running time $\tilde{O}\left(\frac{\bar{N}}{\epsilon}+\frac{\bar{m}}{\epsilon^2}+\frac{\bar{n}}{\epsilon^3}\right)$, where $\tilde{O}(\cdot)$ hides a poly-logarithmic factor.

There has been a recent surge of interest in designing fast or nearly-linear time approximation algorithms for combinatorial optimization problems [11, 12, 9, 15, 34, 6].

Organization. The rest of the paper is organized as follows. In Section 2, we define some elementary notations used across the paper, and describe the result of Young [51] on solving mixed packing and covering LPs, and a template solver for packing LPs over an "easy" polytope. In Sections 3 and 4, we present our results for $R||C_{\max}$ and $R||\sum_j w_jC_j$. Due to the page limit, we leave our algorithms for $P|\text{prec}|\sum_j w_jC_j$ and the two special cases $1|\text{prec}|\sum_j w_jC_j$ and $P|\text{prec}, p_j=1|\sum_j w_jC_j$ to the full version of the paper. The full version also contains other technicalities, such as how to handle the case where input integers are not polynomially bounded, how to reduce problems to the promise versions and how to use the self-balancing binary search tree data structure to run a list scheduling algorithm.

2 Preliminaries

We use bold lowercase letters to denote vectors, and their correspondent italic letters to denote their coordinates. We use bold uppercase letters to denote matrices. $\mathbf{0}$ and $\mathbf{1}$ are used to denote the all-0 and all-1 vectors whose domain can be inferred from the context. Given a template vector $\mathbf{v}$ over some finite domain, and a subset S of the domain, let $v(S) := \sum_{e\in S} v_e$ be the sum of v-values over elements in S.

Given an (undirected) graph $H=(V_H,E_H)$, we use $\delta_H(v), N_H(v), \delta_H(U), N_H(U)$ to respectively denote the sets of incident edges of $v\in V_H$, neighbors of v, edges between the set $U\subseteq V_H$ and $V_H\setminus U$, and vertices in $V_H\setminus U$ with at least one neighbor in U, in the graph H. Given a directed graph $H=(V_H,E_H)$, for every $v\in V_H$, we use $\delta^+_H(v)$ and $\delta^-_H(v)$ to denote the sets of outgoing and incoming edges of v respectively. For every $U\subseteq V_H$, let $\delta^+_H(U) := \{uv\in E_H : u\in U, v\notin U\}$ and $\delta^-_H(U) := \{uv\in E_H : u\notin U, v\in U\}$ be the sets of edges from U to $V_H\setminus U$ and from $V_H\setminus U$ to U respectively. When $H=G$ for the graph G in the context (which can be undirected or directed), we omit the subscript H in the notations.

For cleanness of exposition, we use $\tilde{O}_\epsilon(\cdot)$ to hide factors that are polynomial in $\frac{1}{\epsilon}$ and poly-logarithmic in the size of the input. As we gave the first nearly-linear time algorithms for the studied problems, the hidden factors are small compared to the improvements we make. The final approximation ratios we get have an additive factor of $O(\epsilon)$ (instead of ϵ); but it can be reduced to ϵ if we start from a smaller ϵ. By default, for an (undirected or directed) graph $H=(V_H,E_H)$ we deal with, we assume every vertex is incident to at least one edge so $|E_H|=\Omega(V_H)$. For any $a\in\mathbb{R}$, we define $(a)_+$ as $\max\{a,0\}$.

2.1 Nearly-Linear Time Mixed Packing and Covering LP Solver

A mixed packing and covering LP is an LP of the following form:

$$\text{find } \mathbf{x} \qquad \text{such that} \qquad \mathbf{x} \geq 0, \qquad \mathbf{P}\mathbf{x} \leq \mathbf{1} \qquad \text{and} \qquad \mathbf{C}\mathbf{x} \geq \mathbf{1}, \tag{MPC}$$

where $\mathbf{P} \in \mathbb{R}_{\geq 0}^{\bar{m}_{\mathbf{P}} \times \bar{n}}$ and $\mathbf{C} \in \mathbb{R}_{\geq 0}^{\bar{m}_{\mathbf{C}} \times \bar{n}}$ for some positive integers $\bar{n}, \bar{m}_{\mathbf{P}}, \bar{m}_{\mathbf{C}}$. Let $\bar{m} = \bar{m}_{\mathbf{P}} + \bar{m}_{\mathbf{C}}$ and $\bar{N}$ be the total number of non-zeros in $\mathbf{P}$ and $\mathbf{C}$. Young [51] developed a nearly-linear time algorithm that solves (MPC) approximately:

▶ **Theorem 2.1** ([51]). *Given an instance of* (MPC) *and $\epsilon > 0$, there is an $O\left(\frac{\bar{N}\log \bar{m}}{\epsilon^2}\right)$-time algorithm that either claims* (MPC) *is infeasible, or outputs an $\mathbf{x} \in \mathbb{R}_{\geq 0}^{\bar{n}}$ such that $\mathbf{P}\mathbf{x} \leq (1+\epsilon)\mathbf{1}$ and $\mathbf{C}\mathbf{x} \geq \frac{\mathbf{1}}{1+\epsilon}$.*

2.2 Template Packing LP Solver over a Simple Polytope

In this section, we describe a template MWU-based LP solver for a packing linear program with an additional requirement that the solution is inside an "easy" polytope $\mathcal{Q}$. The framework we describe here is introduced in [10] and later reformulated in [11].

Let $\mathbf{P} \in \mathbb{R}_{\geq 0}^{\bar{m} \times \bar{n}}$ be a non-negative matrix, with $\bar{N}$ non-zero entries. Let $\mathbf{a} \in \mathbb{R}_{\geq 0}^{\bar{n}}$ be a row vector, and $\mathcal{Q} \subseteq \mathbb{R}_{\geq 0}^{\bar{n}}$ be a polytope which is defined by "easy" constraints. We focus on the following linear program:

$$\max\ \mathbf{a}\mathbf{x} \qquad \text{subject to} \qquad \mathbf{x} \in \mathcal{Q} \qquad \text{and} \qquad \mathbf{P}\mathbf{x} \leq \mathbf{1}. \tag{P$_\mathcal{Q}$}$$

Throughout the paper, we make sure all instances of (P$_\mathcal{Q}$) we deal with are feasible.

▶ **Definition 2.2.** *Let $\epsilon \in (0,1), \phi > 0$ be two parameters. An (ϵ, ϕ)-approximate solution to* (P$_\mathcal{Q}$) *is a vector $\mathbf{x} \in \mathcal{Q}$ satisfying $\mathbf{P}\mathbf{x} \leq (1+\epsilon)\mathbf{1}$ and $\mathbf{a}\mathbf{x} \geq \mathbf{a}\mathbf{x}^* - \phi$, where $\mathbf{x}^* \in \mathcal{Q}$ is the optimum solution to* (P$_\mathcal{Q}$).

As a hindsight, we only allow a loss of an additive factor ϕ in the objective function of the LP for $P|\text{prec}|\sum_j w_j C_j$, which will be set to be a polynomially small term. As is typical in a MWU framework, we need to solve the following LP where the constraints $\mathbf{P}\mathbf{x} \leq \mathbf{1}$ are aggregated into one constraint $\mathbf{b}\mathbf{y} \leq 1$, where $\mathbf{b} \in \mathbb{R}_{\geq 0}^{\bar{n}}$ is a row vector:

$$\max\ \mathbf{a}\mathbf{y} \qquad \text{subject to} \qquad \mathbf{y} \in \mathcal{Q} \qquad \text{and} \qquad \mathbf{b}\mathbf{y} \leq 1. \tag{1}$$

Again we guarantee all instances of (1) we encounter are feasible.

▶ **Definition 2.3.** *Let $\epsilon \in (0,1), \phi > 0$ be two parameters. An (ϵ, ϕ)-approximate solution to* (1) *is a vector $\mathbf{y} \in \mathcal{Q}$ satisfying $\mathbf{b}\mathbf{y} \leq 1+\epsilon$ and $\mathbf{a}\mathbf{y} \geq \mathbf{a}\mathbf{y}^* - \phi$, where $\mathbf{y}^*$ is the optimum solution to the LP. An (ϵ, ϕ)-oracle for* (1) *is an algorithm that, given an instance of* (1), *and $\epsilon \in (0,1), \phi > 0$, outputs an (ϵ, ϕ)-approximate solution $\mathbf{y}$ to* (1).

The template LP solver is described in Algorithm 1, where we use $\mathbf{P}_i$ to denote the i-th row vector of $\mathbf{P}$. By our assumption that (P$_\mathcal{Q}$) is feasible, the instance of (1) defined in every execution of Step 3 is also feasible. The performance of the algorithm is summarized in the following theorem.

▶ **Theorem 2.4.** *Algorithm 1 will return an $(O(\epsilon), \phi)$-approximate solution $\mathbf{x}$ to* (P$_\mathcal{Q}$), *within $O(\frac{\bar{m}\log \bar{m}}{\epsilon^2})$ iterations of Loop 2.*

Algorithm 1 LP Solver for $(\mathrm{P}_{\mathcal{Q}})$.

```
Input: an instance of $(\mathrm{P}_{\mathcal{Q}})$, $\epsilon \in (0,1), \phi > 0$, and $(\epsilon,\phi)$-oracle $\mathcal{O}$ for (1)
Output: an $(O(\epsilon),\phi)$-approximate solution $\mathbf{x}$ for $(\mathrm{P}_{\mathcal{Q}})$
1: $t \leftarrow 0, \rho \leftarrow \frac{\ln \bar{m}}{\epsilon^2}, \mathbf{x}^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^{\bar{n}}_{\geq 0}, \mathbf{u}^{(0)} \leftarrow \mathbf{1} \in \mathbb{R}^{\bar{m}}_{\geq 0}$
        ▷ $\mathbf{x}^{(t)}$'s are column vectors and $\mathbf{u}^{(t)}$'s are row vectors
2: while $t < 1$ do
3:     define $\mathbf{b} := \frac{\mathbf{u}^{(t)}}{|\mathbf{u}^{(t)}|}\mathbf{P}$, and run the oracle $\mathcal{O}$ for (1) to obtain an $(\epsilon,\phi)$-approximate solution $\mathbf{y}$ for (1)
4:     $\delta \leftarrow \min\left\{\min_{i\in[\bar{m}]} \frac{1}{\rho \cdot \mathbf{P}_i\mathbf{y}}, 1-t\right\}$
5:     for every $i \in [\bar{m}]$ do $u_i^{(t+\delta)} \leftarrow u_i^{(t)} \cdot \exp\left(\delta\epsilon\rho \cdot \mathbf{P}_i\mathbf{y}\right)$
6:     $\mathbf{x}^{(t+\delta)} \leftarrow \mathbf{x}^{(t)} + \delta\mathbf{y}, t \leftarrow t + \delta$
7: return $\mathbf{x} := \mathbf{x}^{(1)}$
```

Proof. Focus on one iteration of Loop 2. Let t be the value of t at the beginning of the iteration, $\mathbf{y}$ and δ be the $\mathbf{y}$ and δ obtained in Step 3 and 4 in the iteration respectively. Then we have

$$|\mathbf{u}^{(t+\delta)}| = \sum_{i\in[\bar{m}]} u_i^{(t+\delta)} = \sum_{i\in[\bar{m}]} u_i^{(t)} \exp(\delta\epsilon\rho \cdot \mathbf{P}_i\mathbf{y}) \leq \sum_{i\in[\bar{m}]} u_i^{(t)}(1 + (1+\epsilon)\epsilon \cdot \delta\rho \cdot \mathbf{P}_i\mathbf{y})$$
$$= |\mathbf{u}^{(t)}| + (1+\epsilon)\epsilon\delta\rho \cdot \mathbf{u}^{(t)}\mathbf{P}\mathbf{y} \leq |\mathbf{u}^{(t)}| + (1+\epsilon)^2\epsilon\delta\rho \cdot |\mathbf{u}^{(t)}| \leq |\mathbf{u}^{(t)}| \exp((1+\epsilon)^2\epsilon\delta\rho).$$

The inequality in the first line is by that $\delta\rho \cdot \mathbf{P}_i\mathbf{y} \in [0,1]$ for every $i \in [\bar{m}]$ and $e^{\epsilon\theta} \leq 1 + \epsilon\theta + (\epsilon\theta)^2 \leq 1 + \epsilon\theta + \epsilon^2\theta$ for every $\epsilon \in [0,1]$ and $\theta \in [0,1]$. The first inequality in the second line is by that $\frac{\mathbf{u}^{(t)}}{|\mathbf{u}^{(t)}|}\mathbf{P}\mathbf{y} = \mathbf{b}\mathbf{y} \leq 1+\epsilon$.

Combining the inequality over all iterations, we have

$$|\mathbf{u}^{(1)}| \leq |\mathbf{u}^{(0)}| \exp\left((1+\epsilon)^2\epsilon\rho\right) = \bar{m} \cdot \exp\left((1+\epsilon)^2\epsilon\rho\right). \tag{2}$$

For every $i \in [\bar{m}]$, we have $u_i^{(1)} = \exp\left(\epsilon\rho \cdot \mathbf{P}_i\mathbf{x}\right)$, where $\mathbf{x} := \mathbf{x}^{(1)}$ is the returned solution. So, by (2), we have $\exp(\epsilon\rho \cdot \mathbf{P}_i\mathbf{x}) \leq \bar{m} \cdot \exp((1+\epsilon)^2\epsilon\rho)$, which implies $\mathbf{P}_i\mathbf{x} \leq \frac{\ln \bar{m}}{\epsilon\rho} + (1+\epsilon)^2 \leq (1+\epsilon)^2 + \epsilon = 1 + O(\epsilon)$.

In the end $\mathbf{x} = \mathbf{x}^{(1)}$ is a convex combination of vectors $\mathbf{y}$ obtained in all iterations. As each $\mathbf{y}$ is in $\mathcal{Q}$, we have $\mathbf{x} \in \mathcal{Q}$. Moreover, for the instance of (1) in any iteration, $\mathbf{x}^*$ is a valid solution. So, the optimum solution $\mathbf{y}^*$ to the instance of (1) has $\mathbf{a}\mathbf{y}^* \geq \mathbf{a}\mathbf{x}^*$, and the $\mathbf{y}$ returned by the oracle has $\mathbf{a}\mathbf{y} \geq \mathbf{a}\mathbf{y}^* - \phi \geq \mathbf{a}\mathbf{x}^* - \phi$. This implies our final $\mathbf{x}$ has $\mathbf{a}\mathbf{x} \geq \mathbf{a}\mathbf{x}^* - \phi$. Therefore, $\mathbf{x}$ is a $(O(\epsilon),\phi)$-approximate solution to $(\mathrm{P}_{\mathcal{Q}})$.

It remains to bound the number of iterations that Loop 2 can take. In every iteration of loop 2 except for the last one, some i has $\frac{1}{\rho \cdot \mathbf{P}_i\mathbf{y}} = \delta$, i.e., $\delta\epsilon\rho \cdot \mathbf{P}_i\mathbf{y} = \epsilon$. We say u_i is increased fully in the iteration. Notice by (2), each u_i can be increased fully in at most $\frac{\ln\left(\bar{m}\exp((1+\epsilon)^2\epsilon\rho)\right)}{\epsilon} = \frac{\ln \bar{m} + (1+\epsilon)^2\epsilon\rho}{\epsilon} = O\left(\frac{\ln \bar{m}}{\epsilon^2}\right)$ iterations. This bounds the number of iterations by $O\left(\frac{\bar{m}\log \bar{m}}{\epsilon^2}\right)$ as there are $\bar{m}$ different values of i. ◀

For each iteration of loop 2, the steps other than Step 3 takes $O(\bar{N})$ time. Therefore, the running time of Algorithm 1 is $O\left(\frac{\bar{m}\log \bar{m} \cdot \bar{N}}{\epsilon^2}\right)$, plus the time for running the oracle $O\left(\frac{\bar{m}\log \bar{m}}{\epsilon^2}\right)$ times.

3 Unrelated Machine Makespan Minimization

In this section, we give the nearly-linear time $(2+\epsilon)$-approximation algorithm for the unrelated machine makespan minimization problem, i.e, the problem $R||C_{\max}$. Recall that we are given a bipartite graph $G=(M,J,E)$ and a $p_{ij}\in\mathbb{Z}_{>0}$ for every $ij\in E$. Recall that $N(j), N(i), \delta(j)$ and $\delta(i)$ denote the set of neighbors or incident edges of a job $j\in J$ or a machine $i\in M$, in the graph G.

Via a standard technique described in the full version of the paper, we can focus on the following promise version:

- We are given a number $P\geq \mathsf{opt}$, where opt is the optimal makespan of the instance, and our goal is to construct an assignment of makespan at most $(2+O(\epsilon))P$.

For some $ij\in E$ with $p_{ij}>P$, we remove ij from E, as the optimum solution does not use the edge. The following is the natural LP relaxation for the problem:

$$\sum_{j\in N(i)} p_{ij}x_{ij}\leq P, \forall i\in M \quad (3) \qquad \sum_{i\in N(j)} x_{ij}\geq 1, \forall j\in J \quad (4) \qquad x_{ij}\geq 0, \forall ij\in E \quad (5)$$

In the correspondent integer program, $x_{ij}\in\{0,1\}$ for every $ij\in E$ indicates whether the job j is assigned to machine i. (3) requires that the makespan of the schedule to be at most P, (4) requires every job to be scheduled. In the linear program, we replace the requirement that $x_{ij}\in\{0,1\}$ with the non-negativity constraint (5).

By the promise that $P\geq\mathsf{opt}$, the LP is feasible. Therefore, applying Theorem 2.1, we can solve the LP in $\tilde{O}_\epsilon(|E|)$ time to obtain an approximate solution $\mathbf{x}\in[0,1]^E$. By scaling, we can assume (4) holds with equalities, and (3) holds with right side replaced by $(1+O(\epsilon))P$.

To round the solution to an integral assignment in $\tilde{O}_\epsilon(|E|)$-time, we use the grouping idea from [46]: For each machine $i\in M$, we break the fractional jobs assigned to i into groups, each containing $\frac{1}{1+\epsilon}$ fractional jobs. This gives us a bipartite graph H between jobs and groups. Any perfect matching (i.e., a matching covering all jobs J) will give a $(2+O(\epsilon))$-approximation for the makespan problem. In H, every subset $J'\subseteq J$ of jobs has at least $(1+\epsilon)|J'|$ neighbors. The $(1+\epsilon)$-factor allows us to design a $\tilde{O}_\epsilon(|E|)$-time algorithm to find a matching covering all jobs J, as stated in the following lemma:

▶ **Lemma 3.1.** *Assume we are given a bipartite graph $H=(S,T,E_H)$ and $\epsilon>0$ such that $|N_H(S')|\geq(1+\epsilon)|S'|$ for every $S'\subseteq S$. In $O\left(\frac{|E_H|}{\epsilon}\log|S|\right)$-time, we can find a matching in H covering all vertices in S.*

Proof. Let $L=\lfloor\log_{1+\epsilon}|S|\rfloor+1>\log_{1+\epsilon}|S|$. Then we use the shortest-augmenting path algorithm of Hopcroft and Karp [21] to find a matching for which there is no augmenting path of length at most $2L+1$. The running time of the algorithm can be made to $O(|E_H|L)=O(\frac{|E_H|}{\epsilon}\log|S|)$. It remains to show the following lemma:

▶ **Lemma 3.2.** *Let F be a matching in H for which there is no augmenting path of length at most $2L+1$. Then all vertices in S are matched in the matching F.*

Proof. Let $\vec{H}$ be the residual graph of H w.r.t the F: $\vec{H}$ is a directed graph over $S\cup T$, for every edge $st\in E_H$, we have $st\in\vec{H}$, and for every $st\in F$, we have $ts\in\vec{H}$. We say a vertex in S is free if it is unmatched in F. For every integer $\ell\in[0,L]$, define S^ℓ (T^ℓ resp.) to be the set of vertices in S (T, resp.) to which there exists a path in $\vec{H}$ of length *at most* 2ℓ ($2\ell+1$, resp.) from a free vertex. So, we have $S^0\subseteq S^1\subseteq S^2\subseteq\cdots\subseteq S^L$ and $T^0\subseteq T^1\subseteq T^2\subseteq\cdots\subseteq T^L$.

Notice that $T^\ell = N_H(S^\ell)$ for every $\ell \in [0, L]$. So for every $\ell \in [0, L]$, we have $(1+\epsilon)|S^\ell| \leq |T^\ell|$ by the condition of the lemma. All vertices in T^L are matched by our assumption that there are no augmenting paths of length at most $2L+1$. So for every $\ell \in [0, L-1]$, we have $|T^\ell| \leq |S^{\ell+1}|$ as all vertices in T^ℓ are matched to $S^{\ell+1}$.

Combining the two statements gives us $(1+\epsilon)|S^\ell| \leq |S^{\ell+1}|$ for every $\ell \in [0, L-1]$. Thus $|S^L| \geq (1+\epsilon)^L|S^0|$, which contradicts the definition of L and that $|S^0| \geq 1, |S^L| \leq |S|$. ◀

This finishes the proof of Lemma 3.1. ◀

With the lemma, we prove the following theorem using the grouping technique from [39]:

▶ **Theorem 3.3.** *Given $\mathbf{x} \in [0,1]^E$ satisfying $x(\delta(j)) = 1$ for every $j \in J$, and $\epsilon \in (0,1)$, there is an $O\left(\frac{|E|}{\epsilon}\log n\right)$-time algorithm that outputs an assignment $\sigma \in M^J$ of jobs to machines such that $\sigma_j j \in E$ and $x_{\sigma_j j} > 0$ for every $j \in J$, and for every $i \in M$, we have*

$$\sum_{j \in \sigma^{-1}(i)} p_{ij} \leq (1+\epsilon)\sum_{j \in N(i)} p_{ij}x_{ij} + \max_{j \in \sigma^{-1}(i)} p_{ij}. \quad \textit{(Assume the maximum over } \emptyset \textit{ is 0.)}$$

Proof. We construct a bipartite graph $H = (V, J, E_H)$, starting with $V = \emptyset$ and $E_H = \emptyset$. For every machine $i \in M$, we run the following procedure. See Figure 1 for an illustration. (The notations defined in the paragraph depend on i; if a notation does not contain i in the subscript, it will only be used locally, in this paragraph.) Let D_i be the number of jobs j with positive x_{ij} values. Let $j_1, j_2, \cdots, j_{D_i}$ be these jobs j, sorted in non-increasing order of p_{ij}; that is, we have $p_{ij_1} \geq p_{ij_2} \geq \cdots \geq p_{ij_{D_i}}$. For every integer $d \in [0, D_i]$, we define $Z_d = \sum_{d'=1}^{d} x_{ij_{d'}}$. Let $R_i = \lceil (1+\epsilon)Z_{D_i}\rceil = \lceil(1+\epsilon)x(\delta(i))\rceil$. For every $r = 1, 2, 3, \cdots, R_i$, we create a vertex ir and add it to V. We add to E_H an edge between $ir, r \in [R_i]$ and j_d, $d \in [D_i]$ if $(\frac{r-1}{1+\epsilon}, \frac{r}{1+\epsilon}) \cap (Z_{d-1}, Z_d) \neq \emptyset$, and we define $y_{(ir)j_d}$ to be the length of the interval. This finishes the construction of $H = (V, J, E_H)$, along with a vector $\mathbf{y} \in \left(0, \frac{1}{1+\epsilon}\right]^{E_H}$.

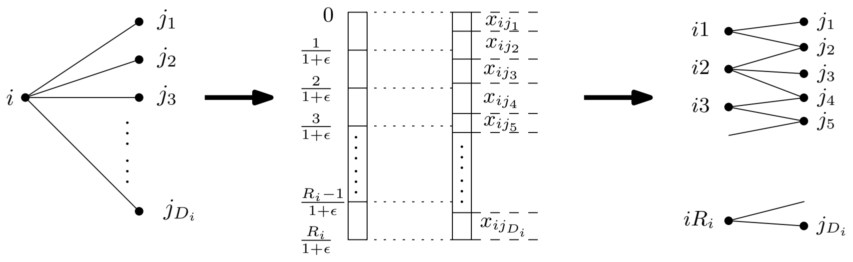

Figure 1 Construction of the H for the machine $i \in M$. In the bipartite graph between $\{i1, i2, \cdots, iD_i\}$ and $\{j_1, j_2, \cdots, j_{D_i}\}$ and there is an edge between j_d and (ir) iff the interval correspondent to j_d intersects the interval $(\frac{r-1}{1+\epsilon}, \frac{r}{1+\epsilon})$.

The number of edges in H for each i is at most $D_i + R_i - 1 \leq |\delta(i)| + (1+\epsilon)x(\delta(i))$. Therefore the total number of edges we created in H is at most $|E| + (1+\epsilon)|J| = O(|E|)$. For every $ij \in E$, we have $\sum_{r:(ir)j \in E_H} y_{(ir)j} = x_{ij}$. This implies that for every $j \in J$, we have $y(\delta_H(j)) = 1$. For every $ir \in V$, we have $y\big(\delta_H(ir)\big) \leq \frac{1}{1+\epsilon}$, and the inequality holds with equality except when $r = R_i$.

For every set $J' \subseteq J$, we have $|N_H(J')| \geq (1+\epsilon)|J'|$, as we can view $\mathbf{y}$ as a fractional matching in H where every $j \in J$ is matched to an extent of 1 and every $ir \in V$ is matched to an extent of at most $\frac{1}{1+\epsilon}$. Then we can use Lemma 3.1 [7] to find a matching in H that

[7] We need to switch the left and right sides when going from the bipartite graph H in Theorem 3.3 to that in Lemma 3.1. That is, we set $S = J$ and $T = V$.

covers all jobs J. The running time of the algorithm is $O\left(\frac{|E_H|}{\epsilon}\log n\right) = O\left(\frac{|E|}{\epsilon}\log n\right)$. The matching gives an assignment $\sigma \in M^J$: If j is matched to ir, then define $\sigma_j = i$. Fix some $i \in M$ with $\sigma^{-1}(i) \neq \emptyset$; we upper bound $\sum_{j\in\sigma^{-1}(i)} p_{ij}$:

$$
\begin{aligned}
\sum_{j\in\sigma^{-1}(i)} p_{ij} \le {} & \max_{j\in\sigma^{-1}(i)} p_{ij} + \sum_{r=2}^{R_i} \max_{j\in N_H(ir)} p_{ij} \\
\le {} & \max_{j\in\sigma^{-1}(i)} p_{ij} + (1+\epsilon)\sum_{r=2}^{R_i} \sum_{j\in N_H(i(r-1))} p_{ij} y_{(i(r-1))j} \\
\le {} & \max_{j\in\sigma^{-1}(i)} p_{ij} + (1+\epsilon)\sum_{r=1}^{R_i} \sum_{j\in N_H(ir)} p_{ij} y_{(ir)j} = \max_{j\in\sigma^{-1}(i)} p_{ij} + (1+\epsilon)\sum_{j\in N(i)} p_{ij} x_{ij}.
\end{aligned}
$$

To see the first inequality, notice that the job j' matched to $i1$ (if it exists) has $p_{ij'} \le \max_{j\in\sigma^{-1}(i)} p_{ij}$, and the job j' matched to each ir, $r \in [2, R_i]$, has $p_{ij'} \le \max_{j\in\delta_H((ir))} p_{ij}$. Consider the second inequality. For every $r \in [2, R_i]$, any $j \in \delta_H(ir)$ and any $j' \in \delta_H(i(r-1))$, we have $p_{ij} \le p_{ij'}$. Moreover, for every $r \in [2, R_i]$, we have $y\big(\delta_H(i(r-1))\big) = \frac{1}{1+\epsilon}$. The inequality in the third line follows from replacing r with $r+1$. The equality holds since for every $ij \in E$ we have $\sum_{r:(ir)j\in E_H} y_{(ir)j} = x_{ij}$. ◀

We can then apply Theorem 3.3 with the solution $\mathbf{x}$ we obtained from solving LP(3-5). Clearly we have $\max_{j\in\sigma^{-1}(i)} p_{ij} \le P$ for every $i \in M$. So, the total load on any machine i is at most $P + (1+\epsilon)\cdot\sum_{j\in N(i)} p_{ij}x_{ij} \le P + (1+\epsilon)\cdot(1+O(\epsilon))P = (2+O(\epsilon))P$, as (3) is satisfied with right side replaced by $(1+O(\epsilon))P$. This finishes the analysis of the algorithm for $R||C_{\max}$ and proves Theorem 1.1.

4 Unrelated Machine Weighted Completion Time Minimization

In this section, we give our nearly-linear time algorithm for $R||\sum_j w_jC_j$, with an approximation ratio of $1.45+\epsilon$, matching the current best ratio of Im and Li [22] achieved in polynomial time. Our result is based on formulating an LP relaxation that is equivalent to the rectangle LP introduced by Li [35]. The new LP relaxation has a nearly-linear size and the mixed packing and covering form; thus it can be solved in nearly-linear time using Theorem 2.1. We describe the rectangle LP (LP(6)), our new LP relaxation (LP(11)) and show their equivalence in Sections 4.1, 4.2 and 4.3 respectively.

In the full version of the paper we show how to construct a solution to LP(6) from one to LP(11) in nearly-linear time, finishing the proof of Theorem 1.2. We also show in the full version that the rounding algorithm of Im and Li can run in nearly-linear time; this finishes the proof of Theorem 1.3. Throughout the section, we assume all processing times are integers bounded by a polynomial of n. The general case is handled in the full version.

4.1 Rectangle LP Relaxation

We describe the rectangle LP relaxation for $R||\sum_j w_jC_j$ introduced by Li [35]. Let $T = \sum_{j\in J}\max_{i\in N(j)} p_{ij}$ so that any schedule will complete by time T. The following is the rectangle LP:

$$
\min \quad \sum_{j\in J} w_j \sum_{i\in N(j), s\in[0,T)} z_{ijs}(s+p_{ij}) \tag{6}
$$

$$\sum_{i\in N(j),s\in[0,T)} z_{ijs} \geq 1 \quad \forall j \in J \qquad (7) \qquad\qquad z_{ijs} = 0 \quad \forall ij \in E, s > T - p_{ij} \quad (9)$$

$$\sum_{j\in N(i),s\in[t-p_{ij},t)} z_{ijs} \leq 1 \quad \forall i \in M, t \in [T] \quad (8) \qquad\qquad z_{ijs} \geq 0 \quad \forall ij \in E, s \in [0,T) \quad (10)$$

In the correspondent integer program, z_{ijs} for every $ij \in E$ and integer $s \in [0,T)$ indicates if job j is scheduled on machine i, with starting time s. The objective gives the weighted completion time of the schedule. (7) requires that every job j is scheduled. (8) requires that at any time on machine i, at most one job is being processed. (9) ensures that no jobs complete after time T. (10) is the non-negativity constraint. Im and Li [22] showed that given a solution $\mathbf{z}$ to LP(6), one can round it to an integral schedule, whose weighted completion time in expectation is at most 1.45 times the value of $\mathbf{z}$.

4.2 A Nearly-Linear Size LP Relaxation

In this section we formulate the relaxation that can be solved in nearly-linear time, and prove its equivalence to LP(6) in Section 4.3. We create a list of time points as follows: $T_0 = 0$, $T_d = \lfloor(1+\epsilon)T_{d-1}\rfloor + 1$ for every integer $d \geq 1$. Define $D = O(\frac{\log n}{\epsilon})$ to be the smallest integer so that $T_D \geq T$. We call $(T_{d-1}, T_d]$ the d-th *window*, and the time points $T_0, T_1, \cdots, T_D$ *window boundaries* (or simply boundaries). Define $\Delta_d = T_d - T_{d-1}$ to be the length of the d-th window.

Let $\eta_d := \lceil \epsilon\Delta_d \rceil$. We partition $(T_{d-1}, T_d]$ into *sub-windows* of length η_d, except that the last sub-window may be shorter. Then $q_d := \left\lceil \frac{\Delta_d}{\eta_d} \right\rceil \leq \frac{1}{\epsilon}$ is the number of sub-windows of $(T_{d-1}, T_d]$. Let $\tau^{(d)}_0 = T_{d-1}, \tau^{(d)}_1, \tau^{(d)}_2, \cdots, \tau^{(d)}_{q_d} = T_d$ be the boundaries of the q_d sub-windows.

We describe the variables in the LP. For every $ij \in E$ and $d \in [D]$ with $p_{ij} \leq \Delta_d$, we introduce a variable x_{ijd}, indicating if j is scheduled on i inside the d-th window. Let S_j and C_j be the starting and completion time of j in the target optimum schedule (which the algorithm does not know). For every $ij \in E, 1 \leq d \leq e < D$, integers $u \in [0, q_d), v \in [0, q_{e+1})$, we *may* introduce a variable y_{ijdeuv} indicating if j is scheduled on i, $S_j \in [\tau^{(d)}_u, \tau^{(d)}_{u+1})$ and $C_j \in (\tau^{(e+1)}_v, \tau^{(e+1)}_{v+1}]$. That means, the scheduling interval $(S_j, C_j]$ of j contains the d'-th window for every $d' \in [d+1, e]$, and a non-empty part of the d-th and $(e+1)$-th windows. u and v approximately give the volumes of j processed in the two windows. It is disjoint from all other windows. As a hindsight, a sub-window is short enough and we can afford to incur an error equaling its length for every window. We only introduce a y-variable if the correspondent event can happen. That is, the following conditions need to be satisfied for the existence of y_{ijdeuv}: $\tau^{(e+1)}_v - \tau^{(d)}_{u+1} + 2 \leq p_{ij} \leq \tau^{(e+1)}_{v+1} - \tau^{(d)}_u$. Notice when $y_{ijdeuv} = 1$, then the scheduling interval $(S_j, C_j]$ of j intersects at least two windows.

For a variable y_{ijdeuv} and an integer $d' \in [D]$, we define

$$Q_{ijdeuvd'} := \begin{cases} 0 & \text{if } d' \leq d-1 \text{ or } d' \geq e+2 \\ \Delta_{d'} & \text{if } d+1 \leq d' \leq e \\ T_d - \tau^{(d)}_{u+1} + 1 & \text{if } d' = d \\ \tau^{(e+1)}_v - T_e + 1 & \text{if } d' = e+1 \end{cases}.$$

Assuming j starts at time $\tau^{(d)}_{u+1} - 1$ and ends at time $\tau^{(e+1)}_v + 1$ on machine i, we have that $Q_{ijdeuvd'}$ is the volume of job j processed in the d'-th window. So, if $y_{ijdeuv} = 1$ in a schedule, then $Q_{ijdeuvd'}$ gives a lower bound on the volume.

We say the quadruple $deuv$ left-covers the pair $d'u'$ if the sub-window $(\tau^{(d')}_{u'-1}, \tau^{(d')}_{u'}]$ is between the sub-windows $(\tau^{(d)}_{u}, \tau^{(d)}_{u+1}]$ (exclusive) and $(\tau^{(e+1)}_{v}, \tau^{(e+1)}_{v+1}]$ (inclusive) in the time horizon, or if $(\tau^{(d')}_{u'-1}, \tau^{(d')}_{u'}] = (\tau^{(d)}_{u}, \tau^{(d)}_{u+1}]$ and $\tau^{(d')}_{u'} - \tau^{(d')}_{u'-1} = 1$. So if $deuv$ left-covers $d'u'$ and $y_{ijdeuv} = 1$, then the scheduling interval of j will surely cover the left-most time unit of the sub-window $(\tau^{(d')}_{u'-1}, \tau^{(d')}_{u'}]$.

With the necessary definitions, we can formulate the LP relaxation as LP(11). For the sake of convenience, we assume if a variable does not exist, then it is not included in a summation.

$$\min \quad \sum_{ijd} w_j \cdot (T_{d-1}+1) \cdot x_{ijd} + \sum_{ijdeuv} w_j \cdot (T_e + 1) \cdot y_{ijdeuv} \tag{11}$$

$$\sum_{id} x_{ijd} + \sum_{ideuv} y_{ijdeuv} \geq 1 \qquad \forall j \in J \tag{12}$$

$$\sum_{\substack{jdeuv\,:\,deuv \\ \text{left-covers } d'u'}} y_{ijdeuv} \leq 1 \qquad \forall i \in M, d' \in [D], u' \in [q_{d'}] \tag{13}$$

$$\sum_{j} p_{ij} \cdot x_{ijd'} + \sum_{jdeuv} Q_{ijdeuvd'} \cdot y_{ijdeuv} \leq \Delta_{d'} \qquad \forall i \in M, d' \in [D] \tag{14}$$

$$\text{all variables are non-negative} \tag{15}$$

Consider the correspondent integer program and an integral schedule. If $x_{ijd} = 1$, then the completion time of j is in $(T_{d-1}, T_d]$. If $y_{ijdeuv} = 1$, then it is in $(T_e, T_{e+1}]$. So, the objective (11) approximates and underestimates the total weighted completion time of the schedule.[8] (12) requires that every job is scheduled: either the scheduling interval of a job j is inside some window, or it overlaps with at least two windows. (13) follows from the definition of $deuv$ left-covering $d'u'$. If $x_{ijd'} = 1$, then the p_{ij} units of job j is processed in the d'-th window on machine i. If $y_{ijdeuv} = 1$, then at least $Q_{ijdeuvd'}$ units is processed. So (14) is valid as the volume of the jobs processed in the d'-th window is at most $\Delta_{d'}$. Therefore, LP(11) is valid, and its value is at most the weighted completion time of the optimum schedule for the instance.

There are at most $D|E|$ x-variables. We then count the number of tuples $ijdeuv$ such that y_{ijdeuv} is a variable in the LP. For fixed $ij \in E, d \in D$ and $u \in [0, q_d)$, there are at most $O(1)$ possibilities for (e, v), as the lengths of sub-windows do not decrease from left to right in the time horizon, except for the last sub-window of each window. Hence the number of y-variables is at most $O\big(\frac{D|E|}{\epsilon}\big) = O\big(\frac{|E|\log n}{\epsilon^2}\big)$.[9] The number of constraints is $O(n + \frac{m|D|}{\epsilon}) = O\big(n + \frac{m \log n}{\epsilon^2}\big)$. The number of non-zeros is at most $O\big(\frac{|E|\log^2 n}{\epsilon^4}\big)$ as each variable appears in at most $O(\frac{D}{\epsilon})$ constraints.

Therefore, by Theorem 2.1, in $O\left(\frac{|E|\log^3 n}{\epsilon^6}\right) = \tilde{O}_\epsilon(|E|)$ time, we can find a solution $(\mathbf{x}, \mathbf{y})$ satisfying the following conditions: Its cost is at most $1+\epsilon$ times that of the optimum solution to the LP, all variables are non-negative, (12) holds with equalities, and (13) and

[8] A more precise estimation for the case $y_{ijdeuv} = 1$ is $\tau^{(e+1)}_{v} + 1$. But the estimation $T_e + 1$ is good enough.

[9] By cutting job lengths p_{ij} by a factor of ϵ, one can reduce the number of y variables to $O\Big(|E|\big(\frac{1}{\epsilon^2} + \frac{\log n}{\epsilon}\big)\Big)$. But we prioritize on giving a clean algorithm, rather than optimizing the poly$(\log n, \frac{1}{\epsilon})$-factor in the running time.

(14) hold with right sides replaced by $1+\epsilon$ and $(1+\epsilon)\Delta_{d'}$ respectively. For convenience, we call such a solution a $(1+\epsilon)$-approximate solution to LP(11); but keep in mind that it may violate (13) and (14) by a factor of $1+\epsilon$.

4.3 Equivalence of LP(11) and LP(6)

We use $\mathsf{lp}_{(6)}$ and $\mathsf{lp}_{(11)}$ to denote the values of LP(6) and LP(11) respectively. It is easy to show that $\mathsf{lp}_{(11)} \leq \mathsf{lp}_{(6)}$, as one can convert a solution to LP(6) into one to LP(11) with smaller or equal value. The following theorem gives the other direction, proving the equivalence of the two LPs up to a $1+O(\epsilon)$ factor:

▶ **Theorem 4.1** (Equivalence of LP(11) and LP(6))**.** *Let $(\mathbf{x},\mathbf{y})$ be a $(1+\epsilon)$-approximate solution to LP(11). Then in $\tilde{O}_\epsilon(|E|)$-time we can find a solution $\mathbf{z}$ to LP(6) except that (8) is only satisfied with the right-side replaced by $1+\epsilon$, such that the following is true for an absolute constant $c \geq 1$, every $ij \in E$ and integer $t \geq 0$:*

$$\sum_{s+p_{ij}>(1+c\epsilon)t} z_{ijs} \leq \sum_{d:T_{d-1}+1>t} x_{ijd} + \sum_{deuv:T_e+1>t} y_{ijdeuv}.$$

In words, for every $ij \in E$, and every time $t \geq 0$, the fraction of job j scheduled on i with completion time after $(1+c\epsilon)t$ in $\mathbf{z}$ is at most the fraction with completion time after t in $(\mathbf{x},\mathbf{y})$. Then, the following corollary is immediate:

▶ **Corollary 4.2.** *Let $(\mathbf{x},\mathbf{y})$ and $\mathbf{z}$ be defined as in Theorem 4.1. Then the value of $\mathbf{z}$ to LP(6) is at most $1+c\epsilon$ times that of $(\mathbf{x},\mathbf{y})$ to LP(11). This implies that the value of $\mathbf{z}$ to LP(6) is at most $(1+c\epsilon)(1+\epsilon)\cdot\mathsf{lp}_{(11)} \leq (1+O(\epsilon))\cdot\mathsf{lp}_{(6)}$.*

To better present the ideas behind the proof, we only show the existence of such a vector $\mathbf{z}$ in this section. That is, we are not concerned with the running time of the algorithm that constructs $\mathbf{z}$. In the full version of the paper we show how $\mathbf{z}$ can be constructed in nearly-linear time.

So the rest of this section is dedicated to proving the existence of $\mathbf{z}$ satisfying the conditions in Theorem 4.1. Till the end, we fix the solution $(\mathbf{x},\mathbf{y})$. We assume all variables in $(\mathbf{x},\mathbf{y})$ have values being integer multiples of $1/\Phi$, and $(1+\epsilon)\Phi$ is an integer, for a large enough integer $\Phi > 0$. We fix a machine $i \in M$ and show how to construct the $\mathbf{z}$ values for this i. We create $(1+\epsilon)\Phi$ *mini-machines*, each serving as $1/\Phi$ fraction of the machine i. We create two types of *mini-jobs*:

- For every variable x_{ijd} with positive value, we create Φx_{ijd} mini-jobs of length p_{ij}; we call them *inside-mini-jobs.* Each such inside-mini-job has an *intended completion time* of $T_{d-1}+1$; this is the estimation used in the objective (11).
- For every variable y_{ijdeuv} with positive value, we create Φy_{ijdeuv} mini-jobs of length $\tau^{(e+1)}_v - \tau^{(d)}_{u+1} + 2$; we call them *cross-mini-jobs.* Notice the length may be smaller than p_{ij}. Similarly, the cross-mini-jobs have an *intended completion time* of T_e+1. We define the *blocking interval* of these mini-jobs to be the union of the sub-windows $(\tau^{(d')}_{u'-1}, \tau^{(d')}_{u'}]$ such that $deuv$ left-covers $d'u'$. This is indeed an interval. As (13) holds with right side replaced by $1+\epsilon$, every time point is covered by blocking intervals of at most $(1+\epsilon)\Phi$ cross-mini-jobs.

Our goal becomes to schedule all the mini-jobs on the $(1+\epsilon)\Phi$ mini-machines integrally, guaranteeing that the completion time of each mini-job is at most $1+5\epsilon$ times its intended completion time (after we extend the lengths of cross-mini-jobs). The construction of the schedule is given in Algorithm 2; recall that we are not concerned with the running time in this proof. The solution $\mathbf{z}$ to LP(6) will be the integral schedule scaled by a factor of $\frac{1}{\Phi}$: z_{ijs} is $\frac{1}{\Phi}$ times the number of mini-jobs for j that start at time s in the schedule.

Algorithm 2 Scheduling of mini-jobs on mini-machines for a machine $i \in M$.

```
1: define a vector $\sigma$ : cross-mini-jobs $\to$ mini-machines, so that for every mini-machine $h$,
   the blocking intervals of $\sigma^{-1}(h)$ are disjoint.
2: for $d' \leftarrow 1$ to $D$ do
3:     for every cross-mini-job $k$ for some variable $y_{ijdeuv}$ with $d \leq d' \leq e+1$ do
4:         $\mathsf{load}_{\sigma_k} \leftarrow \mathsf{load}_{\sigma_k} + Q_{ijdeuvd'}$
5:         if $d' = e+1$ then append $k$ to the mini-machine $\sigma_k$
6:     for every inside-mini-job $k$ for some variable $x_{ijd'}$ do
7:         let $h$ be the mini-machine with the smallest $\mathsf{load}_h$
8:         $\mathsf{load}_h \leftarrow \mathsf{load}_h + p_{ij}$, append $k$ to the mini-machine $h$
9: extend the length of each cross-mini-job for a variable $y_{ijdeuv}$ to $p_{ij}$ in the constructed
   schedule
```

Step 1 of Algorithm 2 is possible since each point is covered by at most $(1+\epsilon)\Phi$ blocking intervals. When we schedule an inside-mini-job k on a mini-machine h, we increase load_h by the length of k (Step 8). The scheduling of a cross-mini-job k for some variable y_{ijdeuv} is done differently. First the mini-machine σ_k for k is pre-defined. Second, we append k to σ_k only in iteration $d' = e+1$ (Step 5), but we add the length of k to load_{σ_k} piece by piece: In iterations $d' = d, d+1, \cdots, e+1$, we increase the load by $Q_{ijdeuvd'}$ (Step 4). Still we ensure that the load to σ_k contributed by k is equal to the length of k. A mini-job for a variable y_{ijdeuv} may have length smaller than the desired length p_{ij}, so in Step 9 we extend these mini-jobs.

▶ **Lemma 4.3.** *At the end of iteration d' of Loop 2, every mini-machine has a load of at most $T_{d'} - 1 + \Delta_{d'}$.*

Proof. There are two types of loads added to mini-machines during iteration d' of Loop 2: those from cross-mini-jobs, and those from inside-mini-jobs. The total load (from both cross- and inside-mini-jobs) added to all mini-machines is at most $(1+\epsilon)\Phi\Delta_{d'}$: it is precisely Φ times the left-side of (14) for the machine i and d', which is at most $(1+\epsilon)\Phi\Delta_{d'}$ as the constraint is violated only by a factor of $1+\epsilon$.

The total load from cross-mini-jobs added to a mini-machine h in iteration d' is at most $\Delta_{d'}$ as the blocking intervals of all mini-jobs in $\sigma^{-1}(h)$ are disjoint. We need to check the case when one mini-job $k \in \sigma^{-1}(h)$ has blocking interval ending at $\tau^{(d')}_{u'}$ and another mini-job $k' \in \sigma^{-1}(h)$ has blocking interval starting at the time. If the length of the sub-window $(\tau^{(d')}_{u'-1}, \tau^{(d')}_{u'}]$ is at least 2, then the statement holds as we only gave 1 unit length to k and k' in this sub-window. If the length is 1, then because we handled the case in a special way in the definition of left-covering, we did not give any length to k' for the sub-window.

With the observations, we can prove the lemma. Before we add an inside-mini-job k for $x_{ijd'}$ to a mini-machine h in iteration d', the total load of all mini-machines is strictly less than $(1+\epsilon)\Phi\sum_{d''=1}^{d'}\Delta_{d''} = (1+\epsilon)\Phi T_{d'}$ (as the length of k has not been added to the loads yet). Therefore $\mathsf{load}_h < T_{d'}$ before we append k to h. After that, we have $\mathsf{load}_h \leq T_{d'} - 1 + p_{ij} \leq T_{d'} - 1 + \Delta_{d'}$.

Assume towards the contradiction that the lemma does not hold and consider the first time when the condition is violated. Assume this is at iteration d', and some mini-machine has a load of at least $T_{d'} + \Delta_{d'}$. This must be due to that we add the loads from cross-mini-jobs to the machine. By our assumption, every mini-machine has a load of at most $T_{d'-1} - 1 + \Delta_{d'-1}$ at the end of iteration $d' - 1$. (A special case is when $d' = 1$; but this can be handled trivially.) As we argued, we add a load of at most $\Delta_{d'}$ from cross-mini-jobs to each mini-machine in iteration d'. Therefore after we add the loads, every mini-machine has a load of at most $T_{d'-1} - 1 + \Delta_{d'-1} + \Delta_{d'} = T_{d'} - 1 + \Delta_{d'-1} \le T_{d'} - 1 + \Delta_{d'}$, a contradiction. ◀

Now we consider how Step 9 changes the completion times. The length of a cross-mini-job for a variable y_{ijdeuv} is increased by at most $\eta_d - 1 + \eta_{e+1} - 1 \le \epsilon\Delta_d + \epsilon\Delta_{e+1} \le 2\epsilon(\Delta_d + \Delta_e)$. For all cross-mini-jobs assigned to the same mini-machine h, the correspondent intervals $\{d, d+1, \cdots, e\}$ are disjoint. Therefore, a mini-job scheduled in iteration d' of Loop 2 is delayed by at most $2\epsilon(\Delta_1 + \Delta_2 + \cdots + \Delta_{d'}) = 2\epsilon T_{d'}$ units time. In the final schedule constructed by Algorithm 2 the completion time of a mini-job scheduled in iteration d is at most

$$T_d - 1 + \Delta_d + 2\epsilon T_d \le T_d - 1 + ((1+\epsilon)T_{d-1} + 1) - T_{d-1} + 2\epsilon T_d$$
$$= T_d + \epsilon T_{d-1} + 2\epsilon T_d \le (1 + 5\epsilon)(T_{d-1} + 1).$$

Setting $c = 5$, Theorem 4.1 follows from that $T_{d-1} + 1$ is the intended completion time of the mini-job.

References

1 Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly-linear time positive LP solver with faster convergence rate. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 229–236, 2015.

2 Måns Alskog. Implementation of a fast approximation algorithm for precedence constrained scheduling. Master's thesis, Linköping University, Applied Mathematics, Faculty of Science and Engineering, 2022.

3 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. `doi:10.4086/toc.2012.v008a006`.

4 Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 453–462, 2009.

5 Nikhil Bansal, Aravind Srinivasan, and Ola Svensson. Lift-and-round to improve weighted completion time on unrelated machines. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing (STOC 2016)*, pages 156–167, 2016.

6 Yair Bartal and Lee-Ad Gottlieb. Near-linear time approximation schemes for steiner tree and forest in low-dimensional spaces. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 1028–1041, 2021. `doi:10.1145/3406325.3451063`.

7 A. Bernstein, M. Gutenberg, and T. Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2021)*, pages 1000–1008, 2021. `doi:10.1109/FOCS52979.2021.00100`.

8 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 775–788, 2020. `doi:10.1145/3357713.3384309`.

9 Chandra Chekuri, Sariel Har-Peled, and Kent Quanrud. Fast lp-based approximations for geometric packing and covering problems. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1019–1038, 2020.

10 Chandra Chekuri, T.S. Jayram, and Jan Vondrak. On multiplicative weight updates for concave and submodular function maximization. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS 2015)*, pages 201–210, 2015.

11 Chandra Chekuri and Kent Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 801–820, 2017.

12 Chandra Chekuri and Kent Quanrud. Fast approximations for metric-tsp via linear programming. *arXiv*, abs/1802.01242, 2018. arXiv:1802.01242.

13 Chandra Chekuri and Kent Quanrud. Randomized MWU for positive LPs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 358–377, 2018.

14 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2022)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

15 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In Sandy Irani, editor, *Proceedings of the 61st Annual IEEE Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1158–1167, 2020. doi:10.1109/FOCS46700.2020.00111.

16 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.

17 Shashwat Garg. Quasi-PTAS for scheduling with precedences using LP hierarchies. In *Proceedings of 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 59:1–59:13, 2018.

18 R. L. Graham. Bounds on multiprocessing timing anomalies. *Siam Journal on Applied Mathematics*, 17(2):416–429, 1969.

19 R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 4:287–326, 1979.

20 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, August 1997.

21 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

22 Sungjin Im and Shi Li. Improved approximations for unrelated machine scheduling. In *Proceedings of the Thirty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 2917–2946, 2023. doi:10.1137/1.9781611977554.ch111.

23 Sungjin Im and Maryam Shadloo. Weighted completion time minimization for unrelated machines via iterative fair contention resolution [extended abstract]. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 2790–2809, 2020.

24 Klaus Jansen and Lars Rohwedder. On the configuration-LP of the restricted assignment problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2670–2678, 2017.

25 Klaus Jansen and Lars Rohwedder. A quasi-polynomial approximation for the restricted assignment problem. *SIAM Journal on Computing*, 49(6):1083–1108, 2020.

26 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 217–226, 2014.

27 Christos Koufogiannakis and N. Young. A nearly linear-time ptas for explicit fractional packing and covering linear programs. *Algorithmica*, 70:648–674, 2013.

28 Christos Koufogiannakis and Neal E. Young. Beating simplex for fractional packing and covering linear programs. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pages 494–504, 2007. doi:10.1109/FOCS.2007.62.

29 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 230–249, 2015.

30 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Proceedings of the Thirty-Second Conference on Learning Theory (COLT 2019)*, pages 2140–2157, 2019.

31 J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, 1978.

32 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

33 Elaine Levey and Thomas Rothvo. A $(1+\epsilon)$-approximation for makespan scheduling with precedence constraints using lp hierarchies. *SIAM Journal on Computing*, 50(3):STOC16–201–STOC16–217, 2021.

34 Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 384–395, 2021. doi:10.1145/3406325.3451114.

35 Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. *SIAM Journal on Computing*, 49(4):FOCS17–409–FOCS17–440, 2020.

36 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC 1993)*, pages 448–457, 1993. doi:10.1145/167088.167211.

37 Alix Munier, Maurice Queyranne, and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *Integer Programming and Combinatorial Optimization (IPCO 1998)*, pages 367–382, 1998.

38 Richard Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1862–1867, 2016.

39 Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. doi:10.1287/moor.20.2.257.

40 Paul Purdom. A transitive closure algorithm. *BIT Numerical Mathematics*, 10:76–94, 1970.

41 Maurice Queyranne and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM J. Comput.*, 35(5):1241–1253, May 2006.

42 Andreas S. Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discret. Math.*, 15(4):450–469, April 2002.

43 Jay Sethuraman and Mark S. Squillante. Optimal scheduling of multiclass parallel machines. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 963–964, 1999.

44 Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, April 1990. doi:10.1145/77600.77620.

45 Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 452–460, 2017. doi:10.1145/3055399.3055501.

46 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(1–3):461–474, February 1993.

47 Martin Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM*, 48(2):206–242, March 2001.

48 Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing (STOC 2010)*, pages 745–754, 2010.

49 Ola Svensson. Santa Claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012.

50 Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1995)*, pages 170–178, 1995.

51 Neal E. Young. Nearly linear-work algorithms for mixed packing/covering and facility-location linear programs, 2014. `arXiv:1407.3015`.