A Faster Combinatorial Approximation Algorithm for Scheduling Unrelated Parallel Machines *

Martin Gairing, Burkhard Monien**, and Andreas Woclaw

Faculty of Computer Science, Electrical Engineering and Mathematics,
University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany.
Email: {gairing,bm,wocland}@uni-paderborn.de

Abstract. We consider the problem of scheduling n independent jobs on m unrelated parallel machines without preemption. Job i takes processing time p_{ij} on machine j, and the total time used by a machine is the sum of the processing times for the jobs assigned to it. The objective is to minimize makespan. The best known approximation algorithms for this problem compute an optimum fractional solution and then use rounding techniques to get an integral 2-approximation.
In this paper we present a combinatorial approximation algorithm that matches this approximation quality. It is much simpler than the previously known algorithms and its running time is better. This is the first time that a combinatorial algorithm always beats the interior point approach for this problem. Our algorithm is a generic minimum cost flow algorithm, without any complex enhancements, tailored to handle unsplittable flow. It pushes unsplittable jobs through a two-layered bipartite generalized network defined by the scheduling problem. In our analysis, we take advantage from addressing the approximation problem directly. In particular, we replace the classical technique of solving the LP-relaxation and rounding afterwards by a completely integral approach. We feel that this approach will be helpful also for other applications.

1 Introduction

We consider the scheduling problem where n independent jobs have to be assigned to a set of m unrelated parallel machines without preemption. Processing job i on machine j takes time p_{ij}. For each machine j, the total time used by ma-

* This work has been partially supported by the DFG-Sonderforschungsbereich 376 Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen, by the European Union within the 6th Framework Programme under contract 001907 (DELIS) and by the DFG Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo)

** Parts of this work were done while the author was visiting Università di Roma *La Sapienza* at Rome and the University of Texas at Dallas chine j is the sum of processing times p_{ij} for the jobs that are assigned to machine j. The makespan of a schedule is the maximum total time used by any machine. The objective is to find a schedule (assignment) that minimizes makespan. This problem has many applications. Typically, they arise in the area of scheduling multiprocessor computers and industrial manufacturing systems (see [18, 28]).

Related Work. There is a large amount of literature on scheduling independent jobs on parallel machines (a collection of several approximation algorithms can be found in [10]). A good deal of these publications concentrate on scheduling jobs on unrelated machines. Horowitz and Sahni [11] presented a (non-polynomial) dynamic programming algorithm to compute a schedule with minimum makespan. Lenstra et al. [17] proved that unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time approximation algorithm for the optimum schedule with approximation factor less than $\frac{3}{2}$. They also presented a polynomial-time 2-approximation algorithm. This algorithm computes an optimal fractional solution and then uses rounding to obtain a schedule for the discrete problem with approximation factor 2. Shmoys and Tardos [23] generalized this technique to obtain the same approximation factor for the generalized assignment problem. They also generalized the rounding technique to hold for any fractional solution.

The fractional unrelated scheduling problem can also be formulated as a *generalized maximum flow* problem, where the network is defined by the scheduling problem and the capacity of some edges, that corresponds to the makespan, is minimized. This generalized maximum flow problem is a special case of *linear programming*. Using techniques of Kapoor and Vaidya [14] and by exploiting the special structure of the problem, an optimum fractional solution can be found with the interior point algorithm of Vaidya [27] in time $O(|E|^{1.5}|V|^2 \log(U))$, where U denotes the maximal p_{ij}.

In contrast to the linear programming methods, the aforementioned generalized maximum flow problem can also be solved with a purely *combinatorial* approach. Here, the makespan minimization is done by binary search. Computing generalized flows has a rich history, going back to Dantzig [2]. The first combinatorial algorithms for the generalized maximum flow problem were exponential time augmenting path algorithms by Jewell [13] and Onaga [19]. Truemper [26] showed that the generalized maximum flow problem and the *minimum cost flow* problem are closely related. More specifically, he transformed a generalized maximum flow problem into some minimum cost flow problem by setting the cost of an edge to be the logarithm of the gain from the generalized maximum flow problem. Goldberg et al. [6] designed the first polynomial-time combinatorial algorithms for the generalized maximum flow problem. Their algorithms were further refined and improved by Goldfarb, Jin and Orlin [7] and later by Radzik [22]. Radzik's algorithm is so far the fastest combinatorial algorithm with a running time of $O(|E|\,|V|(|E| + |V| \log |V|) \log U)$. In order to minimize makespan, this algorithm has to be called at most $O(\log(nU))$ times.

There exist fast fully polynomial-time approximation schemes for computing a fractional solution [4, 12, 20, 21, 25]. Using the rounding technique from [23], this leads to a $(2 + \varepsilon)$-approximation for the discrete problem. The approximation schemes can be divided into those that approximate generalized maximum flows [4, 21, 25] and those that directly address the scheduling problem [12, 20].

Unrelated machine scheduling is a very important problem and many heuristics and exact methods have been proposed. Techniques used here range from combinatorial approaches with partial enumeration to integer programming with branch-and-bound and cutting planes. For a selection we refer to [18, 24, 28] and references therein.

Finding a discrete solution for the unrelated scheduling problem can be formulated as an *unsplittable* generalized maximum flow problem. Several authors [3, 15, 16] have studied the unsplittable flow problem for usual flow networks. Kleinberg [15] formulated the problem of finding a solution with minimum makespan for the restricted scheduling problem as an unsplittable flow problem. Here the restricted scheduling problem is a special case of our problem, in which each job i has some weight w_i, each machine j has some speed s_j and $p_{ij} = \frac{w_i}{s_j}$ or $p_{ij} = \infty$ holds for all i, j. Gairing et al. [5] exploited the special structure of the network, gave a 2-approximation algorithm for the restricted scheduling problem based on preflow-push techniques and also an algorithm for computing a Nash equilibrium for the restricted scheduling problem on identical machines.

Contribution. The algorithm presented in this paper computes an assignment for the unrelated scheduling problem with makespan at most twice the optimum. We prove that a 2-approximative schedule can be computed in $O(m^2 A \log(m) \log(nU))$ time, where A is the number of pairs (i, j) with $p_{ij} \neq \infty$. This is better than the previously known best time bounds of Vaidya's [27] and Radzik's [22] algorithms. In particular, this is the first time that a combinatorial algorithm always beats the interior point approach for this problem.

An essential element of our approximation algorithm is the procedure Unsplittable-Blocking-Flow from [5]. This procedure was designed to solve the unsplittable maximum flow problem in a bipartite network, which is defined by the *restricted* scheduling problem. In this paper the connection to flow is more tenuous. We solve an unsplittable flow problem in a *generalized* bipartite network, which is defined by the *unrelated* scheduling problem. The generalized flow problem can be transformed to a minimum cost flow problem. Our algorithm uses the primal-dual approach combined with a gain scaling technique to obtain a polynomial running time. To compute a flow among the edges with zero reduced cost it uses the procedure Unsplittable-Blocking-Flow from [5] in the inner loop.

Given some candidate value for the makespan, our algorithm finds an approximate solution for the generalized flow problem in the two-layered bipartite network. Throughout execution the algorithm always maintains an integral assignment of jobs to machines. Each assignment defines a partition of the machines into underloaded, medium loaded and overloaded machines. Our overloaded machines are heavily overloaded, that is, their load is at least twice as large as the candidate makespan.

The main idea of our algorithm is to utilize the existence of overloaded machines in conjunction with the fact that we are looking for an approximate integral solution. We use this idea twice. On the one hand this allows us to show an improved lower bound on the makespan of an optimum schedule and thus to overcome the $(1 + \varepsilon)$ error usually induced by the gain scaling technique. On the other hand this is also used to reduce the number of outer loops to $O(m \log m)$, which is the main reason for the substantial running time improvement. Our algorithm is a generic minimum cost flow algorithm without any complex enhancements for generalized flow computation. Overloaded and underloaded machines are treated as sources and sinks, respectively. The height of a node is its minimum distance to a sink. In our algorithm the admissible network, used for the unsplittable maximum flow computation, consists only of edges and nodes which are on shortest paths from overloaded machines with minimum height to underloaded machines. This modification to the primal-dual approach is important to show the improved lower bound on the makespan of an optimum schedule.

Our algorithm is simpler and faster than the previously known algorithms. For the unrelated scheduling problem we have replaced the classical technique, i.e., computing first a fractional solution and rounding afterwards, by a completely integral approach. Our algorithm takes advantage from addressing the approximation problem directly. In particular, this allows us to benefit from an unfavorable preliminary assignment. We feel that this might be helpful also in other applications.

Identifying the connection to flow might be the key for obtaining combinatorial (approximation) algorithms for problems for which solving the LP-relaxation and rounding is currently the (only) alternative. Our techniques and results do not improve upon the approximation factor for the unrelated scheduling problem, however, we expect more exciting improvements for other hard problems.

Comparison of Running Times. We compare the running time of our algorithm with the so far fastest algorithms of Vaidya [27] and Radzik [22]. Both of the former approaches have been designed to solve the fractional generalized maximum flow problem on a graph with node set V and edge set E. Rounding the fractional solution yields the 2-approximation.
Technique and running time for computing a 2-approximative schedule:

- $O(|E|^{1.5}|V|^2 \log(U))$: Interior Point approach for generalized flow problem and rounding [27]
- $O(|E|\,|V|(|E| + |V| \log |V|) \log U \log(nU))$: Combinatorial algorithm for generalized flow problem and rounding [22]
- $O(m^2 A \log(m) \log(nU))$: The integral approach presented in this paper

To compare these bounds, note that in our bipartite network $A = |E| = O(nm)$ and $|V| = n + m$. Our algorithm is linear in A. It clearly outperforms the previous algorithms if $n + m = o(A)$. In the case $A = \Theta(n + m)$ our algorithm is better by a factor of $\Omega(\frac{(n+m)^{0.5}}{\log(n)\log(m)})$ than Vaidya's algorithm and by a factor of $\Omega(\log U)$ faster than Radzik's algorithm. This is the first time that a combinatorial algorithm always beats the interior point approach for this problem.

The heuristics [18, 24, 28] consider instances where $A = \Theta(nm)$. In this case our algorithm outperforms both former approaches by a factor almost linear in n.

The $(1+\varepsilon)$-approximation algorithms for the generalized maximum flow problem in [4, 21, 25] have all running time $\widetilde{O}(\log \varepsilon^{-1}|E|(|E| + |V| \log \log U))$, where the $\widetilde{O}()$ notation hides a factor polylogarithmic in $|V|$. Again, an extra factor of $O(\log(nU))$ is needed for the makespan minimization. This running time is not always better than ours. The fastest approximation scheme that directly addresses the scheduling problem is due to Jansen and Porkolab [12] and has a running time of $O(\varepsilon^{-2}(\log \varepsilon^{-1})mn \min\{m, n \log m\} \log m)$. Clearly, for constant ϵ this algorithm is faster than our algorithm. However, for ϵ in the order of $\frac{1}{m}$ and $\log(U) = O(n)$ their running times become comparable.

Roadmap. The rest of the paper is organized as follows. In Section 2, we introduce notation and model. Section 3.1 presents our approximation algorithm and Section 3.2 shows the analysis.

2 Notation

2.1 The scheduling problem

We consider the problem of scheduling a set J of n independent *jobs* on a set M of m *machines*. The *processing time* of job i on machine j is denoted by p_{ij}. Define the $n \times m$ matrix of processing times $\mathbf{P}$ in the natural way. Throughout the paper we assume that p_{ij} is either an integer or ∞ for all $i \in J$ and $j \in M$. Define $U = \max_{i \in J, j \in M}\{p_{ij} \neq \infty\}$. Furthermore, define A as the number of pairs (i, j) with $p_{ij} \neq \infty$. An *assignment* of jobs to machines is denoted by a function $\alpha : J \mapsto M$. We denote $\alpha(i) = j$ if job i is assigned to machine j. For any assignment α, the *load* δ_j on machine j for a matrix of processing times $\mathbf{P}$ is the sum of processing times for the jobs that are assigned to machine j, thus $\delta_j(\mathbf{P}, \alpha) = \sum_{i \in J, \alpha(i)=j} p_{ij}$. We omit $\mathbf{P}$ in the notation of δ_j if $\mathbf{P}$ is clear from the context.

Define the *makespan* of an assignment α for a processing time matrix $\mathbf{P}$, denoted $\mathsf{Cost}(\mathbf{P}, \alpha)$, as the maximum load on a machine, hence $\mathsf{Cost}(\mathbf{P}, \alpha) = \max_{j \in M} \delta_j(\alpha)$. Associated with a matrix of processing times $\mathbf{P}$ is the *optimum makespan*, which is the least possible makespan of an assignment α, that is $\mathsf{OPT}(\mathbf{P}) = \min_\alpha \mathsf{Cost}(\mathbf{P}, \alpha)$. Following Graham's notation [9], our problem is equivalent to $R||C_{\max}$.

2.2 Generalized Maximum Flows and Minimum Cost Flows

The generalized maximum flow problem is a generalization of the maximum flow problem, where each edge (i, j) has some gain factor μ_{ij}. If f_{ij} units of flow are sent from node i to node j along edge (i, j), then $\mu_{ij} f_{ij}$ units arrive at j. More specifically, let $G = (V, E)$ be a directed graph of the generalized flow problem, $\mu : E \mapsto \mathbb{R}^+$ a gain function, and s and t source and sink node, respectively. Furthermore, there is a capacity function on the edges. A generalized flow $f : E \mapsto \mathbb{R}$ is a function on the edges that satisfies the capacity and antisymmetry constraints on all edges, and the conservation constraints $\sum_{(j,i)\in E}\mu_{ji}f_{ji} - \sum_{(i,j)\in E} f_{ij} = 0$ on all nodes $i \in V \setminus \{s,t\}$. The value of the flow f is defined as the amount of flow into the sink. Among all generalized flows of maximum value, the goal is to find one that minimizes the flow out of the source.

The fractional version of the scheduling problem can be converted into a generalized maximum flow problem [20]. In order to check whether a fractional schedule of length w exists, one can construct a bipartite graph with nodes representing jobs and machines and introduce an edge from machine node i to job node j with gain $1/p_{ij}$. There is a source which is connected to all the machine nodes with edges of unit gain and capacity w, and the job nodes are connected to a sink with edges of unit gain and unit capacity. A generalized flow in this network that results in an excess of n at the sink corresponds to a solution of the fractional scheduling problem. If the maximum excess that can be generated at the sink is below n, then the fractional scheduling problem is infeasible, i.e., the current value of w is too small.

Truemper [26] established a relationship between the generalized maximum flow problem and the minimum cost flow problem. In his construction, he defined the cost for each arc in the minimum cost flow problem as the logarithm of the gain in the generalized maximum flow problem. In order to transform the generalized maximum flow problem to a minimum cost flow problem with integral arc costs, a gain rounding technique can be used (see e.g. [25]). Gains are rounded down to integer powers of some base $b > 1$. The rounded gain of each residual arc (i,j) is defined as $\gamma_{ij} = b^{c_{ij}}$ where $c_{ij} = \lfloor \log_b \mu_{ij} \rfloor$. Antisymmetry is maintained by setting $\gamma_{ij} = 1/\gamma_{ji}$ and $c_{ij} = -c_{ji}$. The cost of arc (i,j) in the resulting minimum cost flow problem equals c_{ij}. Using a potential function $\pi : V \mapsto \mathbb{R}^+$, the reduced costs c^π_{ij} of an arc (i,j) are defined as $c^\pi_{ij} = c_{ij} - \pi(i) + \pi(j)$ (see [1]). The Primal-Dual approach [1] for minimum cost flows can be used to compute a generalized maximum flow (see e.g. [25]). The Primal-Dual approach preserves the *reduced cost optimality condition*, i.e., $c^\pi_{ij} \geq 0$ for each edge (i,j) in the residual network. Because of the rounding, an optimum solution of the minimum cost flow problem gives only a $(1+\epsilon)$-approximation of the generalized (fractional) maximum flow problem. Using techniques from [23], the fractional solution can be transformed to an integral solution. This approach leads to a $(2+\epsilon)$-approximation algorithm for the scheduling problem.

2.3 Our model

We also formulate the scheduling problem as a generalized maximum flow problem. However, we use a different construction as in [20]. We construct a bipartite graph with nodes representing jobs and machines. There is an arc from job node i to machine node j with unit capacity and gain $\mu_{ij} = p_{ij}$ if $p_{ij} \leq w$. The parameter w will be determined by binary search. Each job node i has supply 1. A generalized flow f is a solution to the fractional version of the scheduling problem, if in f all supplies are sent to the machines. In this case, we call f a *feasible* flow. A generalized flow in such a network creates excess on the machine nodes.

An excess on machine j corresponds to the load on machine j. Define $\delta_j(\mathbf{P}, f)$ as the load on machine j under the generalized flow f with gains defined by $\mathbf{P}$. If we require that the supply of each job is sent to exactly one machine, then we get an integral solution to the scheduling problem. In this case, we call f a generalized unsplittable flow and f corresponds to an assignment α, i.e., assigning job i to machine j corresponds to sending one unit of flow along edge (i, j). We are interested in finding a generalized unsplittable flow f such that the maximum excess over all machines is at most $2w$. This is not always possible, however, if we can't find such a flow, we can still derive the lower bound $\mathsf{OPT}(\mathbf{P}) \geq w + 1$.

Following the construction from Section 2.2, we formulate this generalized maximum unsplittable flow problem as a minimum cost flow problem. For the gain rounding, we choose $b = (1 + z)$ where $z = \frac{1}{m}$. If (i, j) is an edge from job node i to machine node j then the cost c_{ij} and the rounded gain γ_{ij} is defined by $c_{ij} = \lfloor \log_b(p_{ij}) \rfloor$, and $\gamma_{ij} = b^{c_{ij}}$. For any path W, we define $\gamma(W) = \prod_{(i,j)\in W} \gamma_{ij}$. In the same way we define $\gamma(K)$ for some cycle K. In the following, denote $\mathbf{C} = (c_{ij})$ and $\mathbf{\Gamma} = (\gamma_{ij})$. In order to solve the minimum cost flow problem we use the well known Primal-Dual approach [1].

For a given assignment α, a positive integer w and a matrix of processing times $\mathbf{P}$, we now define the residual network $G_\alpha(w)$ (Definition 1) and we partition the machines, with respect to their loads, into three subsets (Definition 2).

Definition 1. *Let α be an assignment and $w \in \mathbb{N}$. We define a directed bipartite graph $G_\alpha(w) = (V, E_\alpha(w))$ where $V = M \cup J$ and each machine is represented by a node in M, whereas each job defines a node in J. Furthermore, $E_\alpha = E_\alpha^1 \cup E_\alpha^2$ with $E_\alpha^1 = \{(j, i) : j \in M, i \in J, \alpha(i) = j, p_{ij} \leq w\}$ and $E_\alpha^2 = \{(i, j) : j \in M, i \in J, \alpha(i) \neq j, p_{ij} \leq w\}$.*

Definition 2. *Let $w \in \mathbb{N}$ and α be an assignment. We partition the set of machines M into three subsets:*

$$M^-(\alpha) = \{j : \delta_j(\mathbf{P}, \alpha) \leq w\}$$
$$M^0(\alpha) = \{j : w + 1 \leq \delta_j(\mathbf{P}, \alpha) \leq 2w\}$$
$$M^+(\alpha) = \{j : \delta_j(\mathbf{P}, \alpha) \geq 2w + 1\}$$

In our setting, at each time, nodes from M^- can be interpreted as sink nodes, whereas nodes from M^+ as source nodes.

We now give a lemma that generalizes the path decomposition theorem to generalized flows. The proof of a similar decomposition theorem can be found in [8]. Note, that a fractional generalized flow on a path is defined as a flow that fulfills the flow conservation constraints on the inner nodes. Similarly, a generalized flow on a cycle fulfills the flow conservation constraints on all nodes in the cycle except one.

Lemma 1 (Decomposition theorem). *Let f and g be two generalized feasible flows in $G = (J \cup M, E)$. Then g equals f plus fractional flow: on some directed* *cycles in G_f, and on some directed paths in G_f with end points in M and with the additional property that no end point of some path is also the starting point of some other path.*

2.4 Unsplittable Blocking Flows

Our approximation algorithm will make use of the algorithm Unsplittable-Blocking-Flow introduced in [5]. Unsplittable-Blocking-Flow was designed for a restricted scheduling problem on identical machines. Here, each job i has some weight w_i and is only allowed to use a subset A_i of the machines. This is a special case of the unrelated scheduling problem considered in this paper, where $p_{ij} = w_i$ if $j \in A_i$ and $p_{ij} = \infty$ otherwise. Given an assignment α and an integer w, Unsplittable-Blocking-Flow(α, w) computes an assignment β, where there is no path from $M^+(\beta)$ to $M^-(\beta)$ in $G_\beta(w)$.

We use Unsplittable-Blocking-Flow for arbitrary processing times p_{ij}. In order to make clear that Unsplittable-Blocking-Flow runs on the original processing times (p_{ij}) we include $\mathbf{P}$ in the parameter list. Furthermore, we allow Unsplittable-Blocking-Flow only to reassign jobs according to some graph $G^0_\alpha(w)$, which can be any subgraph of $G_\alpha(w)$. These adaptations do not influence the correctness and the running time of algorithm Unsplittable-Blocking-Flow.

Lemma 2 and Theorem 1 are derived from [5] and state properties of algorithm Unsplittable-Blocking-Flow that are used in the discussion of our approximation algorithm.

Let $G^0_\alpha(w)$ be any subgraph of $G_\alpha(w)$. Let β be the assignment computed by Unsplittable- Blocking-Flow$(\alpha, G^0_\alpha(w), \mathbf{P}, w)$. In this call jobs are reassigned by pushing them through edges of $G^0_\alpha(w)$. We define $G^0_\beta(w)$ as the graph that results from $G^0_\alpha(w)$ after this reassignments.

Lemma 2 ([5, Lemma 4.2]). *Let β be the assignment computed by* Unsplittable-Blocking-Flow*$(\alpha, G^0_\alpha(w), \mathbf{P}, w)$. Then*
(a) $j \in M^-(\alpha) \Rightarrow \delta_j(\mathbf{P}, \beta) \geq \delta_j(\mathbf{P}, \alpha)$
(b) $j \in M^0(\alpha) \Rightarrow w + 1 \leq \delta_j(\mathbf{P}, \beta) \leq 2w$
(c) $j \in M^+(\alpha) \Rightarrow \delta_j(\mathbf{P}, \beta) \leq \delta_j(\mathbf{P}, \alpha)$.

Theorem 1 ([5, Lemma 4.4/Theorem 4.5]). Unsplittable-Blocking-Flow*$(\alpha, G^0_\alpha(w), \mathbf{P}, w)$ takes time $O(mA)$ and computes an assignment β, having the property, that there is no path from $M^+(\beta)$ to $M^-(\beta)$ in $G^0_\beta(w)$.*

3 Approximation Algorithm

We now present our approximation algorithm, Unsplittable-Truemper, which will be used to compute an assignment α where $\mathsf{Cost}(\mathbf{P}, \alpha) \leq 2 \cdot \mathsf{OPT}(\mathbf{P})$. We always maintain an *unsplittable* flow, i.e., an integral solution. We loose a factor of 2 by allowing some gap for the machine loads. The special structure of our algorithm allows us to compensate the error, introduced by the gain scaling technique, by a better lower bound on $\mathsf{OPT}(\mathbf{P})$. We stop the computation as soon as we get this better lower bound. This improves also the running time.

3.1 Algorithm Unsplittable-Truemper

We formulate the scheduling problem as a generalized maximum unsplittable flow problem with rounded gain factors as described in Section 2.2. In order to solve this generalized unsplittable flow problem we use the Primal-Dual approach for computing a minimum cost flow [1]. Our algorithm maintains the reduced cost optimality condition. In our setting this means that it does not create negative cost cycles in the residual network. In order to achieve this, Unsplittable-Truemper iteratively computes a shortest path graph $G^0_\alpha(w)$, which we define below, and uses Unsplittable-Blocking-Flow to compute a blocking flow on this shortest path graph. While the costs in Unsplittable-Truemper refer to the rounded processing times, it operates on the original processing times. It is important to note, that both the costs as well as the original processing times are integer. Because of Theorem 1, there is no path from a machine from M^+ to a machine from M^- in $G^0_\alpha(w)$ after termination of Unsplittable-Blocking-Flow. We stop this procedure, when we can either derive a *good* lower bound on $\mathsf{OPT}(\mathbf{P})$ (see Theorem 2) or we found an assignment α with $M^+ = \emptyset$.

```
UNSPLITTABLE-TRUEMPER(α, P, C, w)
Input: assignment α with each job i assigned to a machine from B(i),
       matrix of processing times P, matrix of edge costs C,
       positive integer w
Output: assignment β

// G_α(w) is the graph corresponding to α and w.
π := 0;
while ∃ machine in M^+ with a path to some machine in M^- in G_α(w)
       and ∀u ∈ M^+ : π(u) < log_b(m)
{
  determine shortest path distances d(·) from all nodes to the set
    of sinks M^- in G_α(w) with respect to the reduced costs c^π_ij;
  update π := π + d;
  define M^+_min as the set of machines from M^+ with minimum distance
    to a node in M^- with respect to the costs c_ij;
  define G^0_α(w) as the admissible graph, consisting only of edges on
    shortest paths from M^+_min to M^- in G_α(w);
  β := UNSPLITTABLE-BLOCKING-FLOW(α, G^0_α(w), P, w);
  update α := β;
 }
return α;
```

We now describe our algorithm in more detail. Unsplittable-Truemper starts with an assignment α. In α, each job $i \in J$ is assigned to some machine $j \in B(i)$, where its processing time is minimum, i.e., $B(i) = \{j \in M : p_{ij} \leq p_{ik}, \forall k \in M\}$. Arc capacities are given by $\mathbf{P}$ whereas arc costs are given by $\mathbf{C}$ (as defined in Section 2). Furthermore, Unsplittable-Truemper gets as input an integer w. Assignment α and integer w define a graph $G_\alpha(w)$ as in Definition 1, and a partition of the machines as in Definition 2. At all times, Unsplittable-Truemper maintains a total assignment, that is all jobs are always assigned to some machine. If a job gets unassigned from a machine, it is immediately assigned to some other machine.

Our algorithm iteratively computes shortest path distances $d(u)$ from each node u to the set of sinks M^-, with respect to the reduced costs c^{π}_{ij}. Then π is updated, such that all arcs on shortest paths have zero reduced costs. For each node $u \in M$, $\pi(u)$ never decreases. After the update of π, $\pi(u)$ holds the minimum distance from u to M^- for each node u with respect to the costs c_{ij}. We define $M^+_{\min}$ as the set of machines from M^+ with minimum distance to a node in M^- with respect to the costs c_{ij}. Note, that $M^+_{\min}$ consists of all machines $u \in M^+$ where $\pi(u)$ is minimum. $G^0_{\alpha}(w)$ is then defined as the admissible graph, consisting only of edges on shortest paths from $M^+_{\min}$ to M^- in $G_{\alpha}(w)$. We will see in Section 3.2 that this is essential for our algorithm. Note, that $G^0_{\alpha}(w)$ consists only of arcs with zero reduced costs. Afterwards, Unsplittable-Blocking-Flow is applied to the admissible graph $G^0_{\alpha}(w)$. It reassigns jobs from the admissible graph, such that after Unsplittable-Blocking-Flow returns, there is no longer a path from a machine in $M^+_{\min}$ to a machine in M^- in the admissible graph $G^0_{\alpha}(w)$. Therefore, $\min\{\pi(u); u \in M^+\}$ increases in the next iteration of the while loop. The residual network $G_{\alpha}(w)$ is then updated accordingly. The while-loop terminates when there exists no machine from M^+ with a path to a machine from M^- in $G_{\alpha}(w)$ or there exists a machine $u \in M^+$ with $\pi(u) \geq \log_b(m)$.

3.2 Analysis

We now analyze the behavior of our algorithm. The main result in this section is Theorem 2. A call of Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w)$ terminates if $M^+(\alpha) = \emptyset$. In this case, we know that $\mathsf{Cost}(\mathbf{P}, \alpha) \leq 2w$. We will see, that we can take also some advantage from an assignment α which is still unfavorable, i.e., for which $M^+(\alpha) \neq \emptyset$ holds.

The reduced cost optimality condition $c^{\pi}_{ij} \geq 0$ holds for all $(i,j) \in E_{\alpha}(w)$ during the whole computation. It implies $\gamma(K) \geq 1$ for each cycle K in $G_{\alpha}(w)$. This property does not necessarily hold for every path. Lemma 3 is of crucial importance in our analysis. It shows that $\gamma(W) \geq 1$ holds for every path W connecting some node from $M^+(\alpha)$ to any other node from M in $G_{\alpha}(w)$. For proving this result, we need that $G^0_{\alpha}(w)$ was defined only by shortest paths from nodes in $M^+_{\min}$ to nodes in M^-.

Lemma 3. Unsplittable-Truemper *maintains the property, that for each path W in $G_{\alpha}(w)$ from any machine in M^+ to any other machine in M, we have $\gamma(W) \geq 1$.*

The following lemma will be used to derive a lower bound on $\mathsf{OPT}(\mathbf{P})$.

Lemma 4. *Let $(G, \mathbf{\Gamma})$ denote a generalized maximum unsplittable flow problem defined by network G and matrix of processing times $\mathbf{\Gamma}$. Let f be a generalized feasible unsplittable flow in $(G, \mathbf{\Gamma})$, and let $s, t \in \mathbb{R}^+$. Suppose $\forall u \in M : \delta_u(\mathbf{\Gamma}, f) \geq$*

s, and $\exists \hat{u} \in M : \delta_{\hat{u}}(\mathbf{\Gamma}, f) \geq s + t$, and for each cycle K in G_f, $\gamma(K) \geq 1$. If on every path W in G_f from $\hat{u}$ to any other machine $u \in M$, $\gamma(W) \geq 1$, then $\mathsf{OPT}(\mathbf{\Gamma}) \geq s + \frac{t}{m}$.

Theorem 2. Unsplittable-Truemper *takes time $O(m^2 A \log(m))$. Furthermore, if* Unsplittable-Truemper*$(\alpha, \mathbf{P}, \mathbf{C}, w)$ terminates with $M^+ \neq \emptyset$ then $\mathsf{OPT}(\mathbf{P}) \geq w + 1$.*

We will now show how to use Unsplittable-Truemper to approximate a schedule with minimum makespan. We do series of calls to Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w)$ where, by a binary search on $w \in [1, nU]$, we identify the smallest w such that a call to Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w)$ returns an assignment with $M^+ = \emptyset$. Afterwards we have identified a parameter w, such that Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w)$ returns an assignment where $M^+ \neq \emptyset$ and Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w + 1)$ returns with $M^+ = \emptyset$.

Theorem 3. Unsplittable-Truemper *can be used to compute a schedule α with $\mathsf{Cost}(\mathbf{P}, \alpha) \leq 2 \cdot \mathsf{OPT}(\mathbf{P})$ in time $O(m^2 A \log(m) \log(nU))$.*

Proof. We use Unsplittable-Truemper as described above. Let β_1 be the assignment returned by Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w)$ where $M^+ \neq \emptyset$. Let β_2 be the assignment returned by Unsplittable-Truemper$(\alpha, \mathbf{P}, \mathbf{C}, w+1)$ where $M^+ = \emptyset$. From β_1 we follow by Theorem 2 that $\mathsf{OPT}(\mathbf{P}) \geq w+1$ and in β_2 we have $\mathsf{Cost}(\mathbf{P}, \beta_2) \leq 2(w + 1)$. Thus, $\mathsf{Cost}(\mathbf{P}, \beta_2) \leq 2 \cdot \mathsf{OPT}(\mathbf{P})$. It remains to show the running time of $O(m^2 A \log(m) \log(nU))$. Due to Theorem 2, one call to Unsplittable-Truemper takes time $O(m^2 A \log(m))$. The binary search contributes a factor $\log(nU)$. This completes the proof of the theorem. □

Acknowledgments. We would like to thank Thomas Lücking for many fruitful discussions and helpful comments.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.
2. G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New York, 1963.
3. Y. Dinitz, N. Garg, and M.X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.
4. L. Fleischer and K. D. Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91(2):215–238, 2002.
5. M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing Nash equilibria for scheduling on restricted parallel links. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC'04)*, pages 613–622, 2004.

6. A.V. Goldberg, S.A. Plotkin, and E. Tardos. Combinatorial algorithms for the generalized circulation problem. *Math. of Operations Research*, 16:351–379, 1991.
7. D. Goldfarb, Z. Jin, and J.B. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Math. of Operations Research*, 22:793–802, 1997.
8. M. Gondran and M. Minoux. *Graphs and Algorithms.* Wiley, 1984.
9. R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
10. D.S. Hochbaum. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Co., 1996.
11. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.
12. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Math. of Operations Research*, 26(2):324–338, 2001.
13. W.S. Jewell. Optimal flow through networks with gains. *Operations Research*, 10:476–499, 1962.
14. S. Kapoor and P.M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC'86)*, pages 147–159, 1986.
15. J. Kleinberg. Single-source unsplittable flow. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 68–77, 1996.
16. S.G. Kolliopoulos and C. Stein. Approximation algorithms for single-source unsplittable flow. *SIAM Journal on Computing*, 31:919–946, 2002.
17. J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
18. E. Mokotoff and P. Chrétienne. A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*, 141:515–525, 2002.
19. K. Onaga. Dynamic programming of optimum flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, 13:308–327, 1966.
20. S.A. Plotkin, D.B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. of Operations Research*, 20(2):257–301, 1995.
21. T. Radzik. Faster algorithms for the generalized network flow problem. *Math. of Operations Research*, 23:69–100, 1998.
22. T. Radzik. Improving time bounds on maximum generalised flow computations by contracting the network. *Theoretical Computer Science*, 312(1):75–97, 2004.
23. D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.
24. F. Sourd. Scheduling tasks on unrelated machines: Large neighborhood improvement procedures. *Journal of Heuristics*, 7:519–531, 2001.
25. E. Tardos and K. D. Wayne. Simple generalized maximum flow algorithms. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO'98)*, pages 310–324, 1998.
26. K. Truemper. On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, 32(2):450–456, 1977.
27. P.M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 332–337, 1989.
28. S. L. van de Velde. Duality-based algorithms for scheduling unrelated parallel machines. *ORSA Journal on Computing*, 5(2):182–205, 1993.