Technische Universität Berlin

FAKULTÄT II
MATHEMATIK UND
NATURWISSENSCHAFTEN
Institut für Mathematik

Scheduling AND/OR-Networks on Identical Parallel Machines

by

Thomas Erlebach Vanessa Kääb
Rolf H. Möhring

No. 047-2003

Scheduling AND/OR-Networks on Identical Parallel Machines

Thomas Erlebach* Vanessa Kääb†‡ Rolf H. Möhring†§

December 2003

Abstract

Scheduling precedence constrained jobs on identical parallel machines is a well investigated problem with many applications. AND/OR-networks constitute a useful generalization of standard precedence constraints where certain jobs can be executed as soon as at least one of their direct predecessors is completed. For the problem of scheduling AND/OR-networks on parallel machines, we present a 2-approximation algorithm for the objective of minimizing the makespan. The main idea of the algorithm is to transform the AND/OR constraints into standard constraints. For the objective of minimizing the total weighted completion time on one machine, scheduling AND/OR-networks is as hard to approximate as LABEL COVER. We show that list scheduling with shortest processing time rule is an $O(\sqrt{n})$-approximation for unit weights on one machine and an n-approximation for arbitrary weights.

1 Introduction

Scheduling precedence constrained jobs on identical parallel machines is a well investigated problem with many applications. A precedence constraint of the form $i \prec j$ means that job j can be started only after the completion of job i. If there are several jobs i with $i \prec j$, the job j can be started only when *all* of these jobs i are completed (AND-constraint). A natural and useful generalization of standard precedence constraints are AND/OR precedence constraints, represented by AND/OR-networks. In addition to standard constraints, they allow to specify that a node can begin execution as soon as *at least one* of its direct predecessors is completed (OR-constraint). AND/OR-networks arise in many applications, for example in resource-constrained project scheduling [20] and in assembly/disassembly sequencing [10]. In the latter application, a given product may have to be disassembled. Certain components can be removed only after the removal of other components, leading to standard AND-constraints. However, it might also be possible to remove a component from one of several geometric directions, assuming that other components blocking *this direction* have been removed beforehand. This case naturally leads to OR-constraints. Therefore, the different possibilities how to reach a component can be suitably modeled by AND/OR precedence constraints.

If a given set of jobs with AND/OR precedence constraints has to be executed by a bounded number of identical machines (e.g., assembly lines, workers, etc.), interesting optimization problems arise. Natural objectives are the makespan (the completion time of the job that finishes last), the total completion time (the sum of the completion times of all jobs), and the total weighted completion time (where the completion time of each job is multiplied with the weight of the job). While these problems have been studied intensively for standard precedence constraints, only little is known about scheduling AND/OR-networks on one or several machines.

The makespan is certainly one of the most relevant objective functions for many applications. No matter which aims are pursued in a real project, the overall completion time of the project in general plays an important role. The total completion time is proportional to the average completion time and is therefore a useful criterion in applications where the completion of each individual job brings a certain benefit. The total weighted completion time is a natural variation where the importance of completing a job early can be specified by its weight. The total

*ETH Zürich, Computer Engineering and Networks Laboratory (TIK), Gloriastrasse 35, CH-8092 Zürich, Switzerland, Fax +(41)-1-632-1036, erlebach@tik.ee.ethz.ch, http://www.tik.ee.ethz.ch/~erlebach.

†Technische Universität Berlin, Fakultät II, Institut für Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D-10623 Berlin, Germany, Fax +(49)-30-314-25191, {kaeaeb, moehring}@math.tu-berlin.de, http://www.math.tu-berlin.de/coga.

‡Former member of the European Graduate Program 'Combinatorics, Geometry, and Computation', supported by the Deutsche Forschungsgemeinschaft (DFG) under grant GRK 588/2.

§The author was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Mo 446/3-4.

weighted completion time is also a useful objective function in compiler optimization, where fast algorithms are needed to exploit the parallelism provided by pipelined, super-scalar, and very-long instruction word architectures [14, 23]. Profile-driven code optimization with the total weighted completion time as a relevant cost measure is also considered in [2].

1.1 Known Results

Scheduling with the makespan objective is trivial on one machine if the jobs are not restricted at all, but also if standard or AND/OR precedence constraints are imposed among the jobs. The problem on identical parallel machines is NP-complete even without precedence constraints and with just two machines. Nevertheless, Graham's list scheduling provides a simple 2-approximation for scheduling precedence constrained jobs on any number of parallel machines [11]. Gillies and Liu [8] present a 2-approximation for scheduling jobs of an acyclic AND/OR-network. Their algorithm first transforms the AND/OR-network into a standard precedence graph and then applies Graham's list scheduling.

Minimizing the total weighted completion time is a non-trivial problem even on a single machine. If there are no precedence constraints among the jobs, it is well known that scheduling the jobs according to *Smith's rule* [22] (in order of non-decreasing processing time over weight ratio) yields an optimal solution. For the total weighted completion time of standard precedence constrained jobs on one machine, 2-approximation algorithms have been obtained with various techniques [13, 4, 3, 19]. For the problem with identical parallel machines, a 4-approximation algorithm was presented in [21]. Unfortunately, it seems that the techniques used to obtain these algorithms resist any application to AND/OR precedence constraints.

1.2 Our Results

In this paper, we study scheduling problems with AND/OR precedence constrained jobs on one or several machines. For the problem of minimizing the makespan on identical parallel machines, we extend the algorithm due to Gillies and Liu [8] to general feasible AND/OR-networks (which may contain cycles) in order to obtain a 2-approximation algorithm. Then we consider the problem of minimizing the total weighted completion time on a single machine. For the special case with unit weights (i.e., total completion time), we prove that list scheduling with *shortest processing time rule* is an $O(\sqrt{n})$-approximation, where n is the number of jobs. This bound is tight. The case with arbitrary weights seems to be harder to approximate. We observe that a reduction from LABEL COVER proposed by Goldwasser and Motwani in [9, 10] together with an improved inapproximability result for LABEL COVER by Dinur and Safra [5] shows that minimizing the total weighted completion time to within a factor of $2^{\log^{1-1/\log\log^c n} n}$ of the optimum is NP-hard for any $c < 1/2$. We prove that list scheduling with shortest processing time rule is a simple n-approximation for the problem and that this bound is again tight.

2 Preliminaries

2.1 Problem Description

A scheduling instance consists of a set of n *jobs* $V = \{1, 2, \ldots, n\}$. With each job $j \in V$ we associate a (strictly) positive *processing time* p_j and a non-negative *weight* ω_j. In addition to the jobs, a number m of identical parallel *machines* will be given. The machines all run with the same speed and can process any job. Of course, at any point in time, each machine can process only one job and every job can only be processed by one machine. We will restrict to the non-preemptive case. Once a job is started on a certain machine, it will be finished on that machine without any interruption.

The jobs are subject to two different classes of constraints. On one hand, standard *precedence constraints* represented by a directed acyclic graph $G = (V(G), E(G))$ are given. An edge $(i, j) \in E(G)$ represents the constraint that j has to wait for the completion of i. In a feasible realization of such a problem, all jobs have to be executed in accordance to the partial order induced by G. Each job $j \in V$ has to wait for the completion of all its predecessors in the partial order and thus will also be called an *AND-node*. On the other hand, we also allow for precedence relations of the form that a job j has to wait for the completion of only one predecessor. Those restrictions cannot be captured by the classical precedence constraints described above. The model of standard precedence constraints can be generalized by the introduction of a set W of waiting conditions. A *waiting condition* is an ordered pair $w = (X, j)$, where $X \subseteq V$ is a set of jobs and $j \in V \setminus X$ is the *waiting job*. The waiting job j can be processed as soon as one of the jobs in X has been completed. The standard precedence constraints together with the waiting conditions can be represented by an *AND/OR-network* $N = (V \cup W, E)$ in the following way: for every waiting condition $w = (X, j) \in W$, we introduce an *OR-node*, which we will denote by w again. For every $x \in X$, we introduce a directed edge (x, w). In addition, there is a directed edge (w, j) for the waiting job j. To model the required constraints correctly, we impose the rule that an OR-node w can be scheduled as soon as any of its predecessors $x \in X$ is completed. An OR-node $w \in W$ can be considered as a pure dummy node with processing time $p_w = 0$ and weight $\omega_w = 0$. For convenience, we assume N to have a common start vertex, the *source* s, and a common end vertex, the *sink* t, with $p_s = p_t = 0$ and $\omega_s = \omega_t = 0$. For an illustration, an AND/OR-network is depicted in the left part of Figure 1, where node w_3 represents the waiting condition $(\{5, 6\}, 7)$, for example.

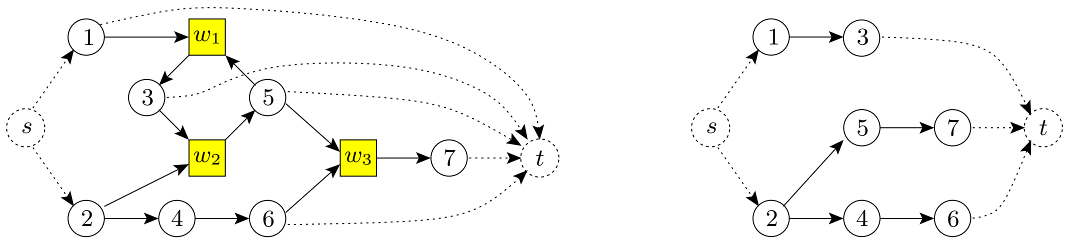

Figure 1: An AND/OR-network N on the left and a realization of N on the right side. AND-nodes are drawn as circles and OR-nodes as shades squares.

In contrast to standard precedence constraints, an AND/OR-network may contain cycles and be feasible at the same time. Remember that the AND/OR-network $N = (V \cup W, E)$ contains the precedence digraph G, as a subgraph. A *realization* of an AND/OR-network N is a partial order $R = (V, <_R)$ which is an extension of G, that is $i <_R j$ for each $(i, j) \in E$ with $i, j \in V$, and

for each $w = (X, j) \in W$, there exists $i \in X$ with $i <_R j$.

A linear or total order $L = (V, <_L)$, which is a realization of N, is called *linear realization*. In Figure 1 a realization of the AND/OR-network N on the left is depicted on the right hand side. A total order is given by $L = 1, 2, 3, 4, 5, 6, 7$, for example. An AND/OR-network N is called *feasible* if it has a realization R. We refer to [15, 20] for additional characterizations of feasible AND/OR-networks and a linear time algorithm to check for feasibility.

For a given instance $(N = (V \cup W, E), p, \omega, m)$, a feasible *schedule* is a non-negative vector of start times $S = (S_1, \dots, S_n)$ for the jobs and waiting conditions such that

1. $S_j \geq \max\{S_v + p_v \mid (v, j) \in E\}$ for all $j \in V$ (*AND-constraints*)
2. $S_w \geq \min\{S_v + p_v \mid (v, w) \in E\}$ for all $w \in W$ (*OR-constraints*)
3. $|\{j \in V \mid S_j \leq t < S_j + p_j\}| \leq m$ at any time $t \geq 0$ (*machine constraints*)

For a given schedule S, the corresponding completion time vector C is defined by $C_j = S_j + p_j$. The makespan of schedule S is denoted by $C_{\max}(S) = \max_{j \in V} C_j = C_t$. The total completion time of S is $\sum_{j \in V} C_j$ and the total weighted completion time is $\sum_{j \in V} \omega_j C_j$.

Let us consider the problem without machine constraints for the moment, that is $m = \infty$. By definition, $S = (\infty, \dots, \infty)$ is a feasible schedule and if $S = (S_1, \dots, S_n)$ and $S' = (S'_1, \dots, S'_n)$ are feasible schedules, then also the component wise minimal schedule $S'' = (\min\{S_1, S'_1\}, \dots, \min\{S_n, S'_n\})$. It follows that there exists a (unique) component wise minimal schedule, the so-called *earliest start schedule* (*ES*). We assume strictly positive job processing times, thus an earliest start schedule can be computed by a modification of Dijkstra's shortest path algorithm in polynomial time. For further details we refer to [15]. Similar algorithms have been presented before in [16, 1, 20]. The earliest start schedule *ES* has the following property. For every job $j \in V$, the inequality $S_j \geq \max\{S_v + p_v \mid (v, j) \in E\}$ is fulfilled with equality for at least one edge $(v, j) \in E$ as otherwise j could start earlier. For the same reason, for each OR-node $w \in W$, the inequality $S_w \geq \min\{S_v + p_v \mid (v, w) \in E\}$ is also fulfilled with equality for at least one edge. We call those edges (v, j) and (v, w) for which equality holds *tight*.

In the earliest start schedule, every job is started as early as possible without violating any constraint. Therefore it also finishes as early as possible. For the problem without machine constraints it follows that the earliest start schedule ES is an optimal schedule for all three objective functions, the makespan, the total completion time, and the total weighted completion time. In the following we will always denote an *optimal schedule* for a given problem by $S^* = (S_1^*, \ldots, S_n^*)$.

To describe the problem under consideration quickly we will use the $\alpha|\beta|\gamma$-notation of Lawler, Lenstra, Rinnooy Kan, and Shmoys [18]. The first entry, α, specifies the machine environment, β describes the job characteristics, where $\beta = \textit{ao-prec}$ denotes AND/OR precedence constraints, and γ specifies the objective function. We refer to [12, 18] for a detailed description. As an example, the problem of scheduling jobs with unit processing times and AND/OR precedence constraints on one machine with the objective to minimize the total completion time is specified by $1|\textit{ao-prec}, p_j = 1|\sum C_j$.

2.2 List Scheduling

One of the most basic and intuitive approaches to tackle a given scheduling problem is *list scheduling*. The idea is to order the jobs according to some priority rule, which gives an ordered list. Then the jobs are scheduled according to this priority list. Whenever a machine is free, the first job in the list that has not been scheduled yet and is available at that time is assigned to be processed by the free machine. List scheduling can easily be implemented to run in polynomial time. According to [18], this algorithm has been presented first by Graham [11] and we will also refer to it as *Graham's list scheduling algorithm*.

In the literature, list scheduling is also called a priority-driven heuristic, according to the priority list that guides the order in which the jobs are scheduled. In most of the algorithms we will consider, list scheduling will be a basic component. Priority driven heuristics never intentionally leave machines idle. A machine is only left idle if there are currently no jobs available. This means that for every job in the list, at least one predecessor is still in process. Recursing this argument gives a well known fact for standard precedence constraints stated in the next lemma. Let $G = (V, E)$ be a standard (acyclic) precedence graph with a strictly positive processing time vector p and $u, v \in V$ two nodes such that there exists a u-v-path in G. The length of a longest path from u to v shall be denoted by $\ell_{uv}^{\max}$, where $\ell_{uv}^{\max} = p_v$ if $u = v$ and $\ell_{uv}^{\max} = p_v + \max\{\ell_{ux}^{\max} \mid (x, v) \in E$ and there exists a u-x-path in $G\}$ if $u \neq v$.

Lemma 2.1 (Graham [11]). *In any list schedule for a set of precedence constrained jobs, there is an s-t-path of jobs that is executed during all periods when some machine is idle, and the length of this path is not longer than the makespan of an optimal schedule.*

The big advantage of Graham's list scheduling is that it is easy, both to understand and to implement, and that it is applicable for a wide range of problems. We now turn to our scheduling problems with AND/OR precedence constraints.

3 The Makespan on Identical Parallel Machines

Minimizing the makespan of a set of precedence constrained jobs on one machine like an assembly line, a worker or one processor, is trivial. If the jobs are not restricted at all, we can simply schedule them in any order without interruption and idle time in between. If standard precedence constraints are involved, we can process the jobs in order of a linear extension of the partial order G representing the precedences. In the case of AND/OR precedence constraints, the solution is equally simple, the jobs can be executed in the order of a linear realization of the AND/OR-network. In every case, the makespan is equal to the sum of the processing times of the jobs, assuming that the constraints are feasible of course.

Now consider the case of scheduling a set of precedence constrained jobs on m identical parallel machines. The problem $Pm||C_{\max}$ is already NP-complete for $m = 2$, but can be solved in pseudo-polynomial time for any fixed number m of machines. For an arbitrary number of machines, that is $P||C_{\max}$, the problem is NP-complete in the strong sense [6].

Nevertheless, Graham's list scheduling (GLS) performs provably well for this problem. Let S^{GLS} denote the schedule produced by list scheduling for $P||C_{\max}$ on m machines. Graham [11] proved that, for any instance,

$$C_{\max}(S^{GLS}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*).$$

Algorithm 1: Earliest Start List Schedule

Input: AND/OR-network $N = (V \cup W, E)$, number $m \geq 1$ of machines, processing time vector $p > 0$

Output: schedule $S = (S_1, \dots, S_n)$

compute earliest start schedule ES for (N, p) (no machine constraints);
let $G = (V, E(G))$ be the subgraph of N induced by V;
foreach *OR-node* $w = (X, j) \in W$ **do**
  for *exactly one* $x \in X$ *with* $ES_x + p_x = ES_w$ **do**
    add edge (x, j) to $E(G)$;

let L be a linear extension of the resulting AND-graph;
apply List Scheduling to the AND-graph G with p and L to obtain the schedule S;
return S;

He also showed that this performance guarantee is not affected by precedence constraints. Thus list scheduling is a 2-approximation for $P|prec|C_{\max}$.

As a generalization of standard precedence constraints, the problem of minimizing the makespan of AND/OR constrained jobs on identical parallel machines, $P|ao\text{-}prec|C_{\max}$, is NP-complete. Fortunately, the problem with AND/OR precedence constraints can be reduced to the problem with standard precedence constraints preserving the approximation guarantee. Gillies and Liu [8] present a 2-approximation algorithm for acyclic AND/OR-networks. The basic idea of the so-called *Minimum Path Heuristic* in [8] is to first change the AND/OR-network into an AND-only network, that is a standard precedence digraph, and then apply Graham's list scheduling. For this transformation, Gillies and Liu use a recursive argument which minimises the longest path to each OR-node provided that only AND-nodes precede the OR-node. This construction fails in the presence of cycles. We present an algorithm similar to the Minimum Path Heuristic which is applicable to AND/OR-networks containing cycles.

The strategy of our *Earliest Start List Schedule* presented in Algorithm 1 is to fix the predecessor of an OR-node to one of the tight (with respect to the earliest start schedule) direct predecessors. Then the OR-node can be removed by making the chosen predecessor a direct predecessor of the immediate successor of the OR-node. Note that this precedence graph G is a realization of N. In addition it minimizes the longest path to each OR-node, respectively its waiting job, among all realizations. Every step of the Earliest Start List Schedule (ESL) can be executed in polynomial time, thus the Earliest Start List Schedule can be implemented to run in polynomial time. Together with Lemma 2.1 by Graham and the optimality of the earliest start schedule with respect to the makespan without machine constraints, we are able to prove an approximation guarantee of 2.

Theorem 3.1. *Let S^{ESL} denote the schedule computed by the Earliest Start List Schedule and S^* an optimal schedule for an instance of $P|ao\text{-}prec|C_{\max}$. Then*

$$C_{\max}(S^{ESL}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*).$$

Moreover, this bound is tight.

Proof. Consider the schedule S^{ESL} and its makespan $C_{\max}(S^{ESL})$ computed by the Earliest Start List Schedule. At any time $0 < t \leq C_{\max}(S^{ESL})$ either all machines are busy or one or more machines are idle. Accordingly, we can divide the time between 0 and $C_{\max}(S^{ESL})$ into busy periods and idle periods. We denote the total length of all busy periods by T_b and the total length of all idle periods by T_i, thus

$$C_{\max}(S^{ESL}) = T_b + T_i.$$

For the total length of idle periods, $T_i \leq \ell_{st}^{\max}(G)$ by Lemma 2.1. From the construction of graph G we get that $\ell_{st}^{\max}(G) = C_{\max}(ES)$, where ES is the earliest start schedule of (N, p) without machine constraints computed in the first step of the algorithm. We know that the earliest start schedule achieves the optimal makespan for an instance (N, p) and therefore is a lower bound on $C_{\max}(S^*)$. Together this yields

$$T_i \leq C_{\max}(S^*).$$

Additionally, we can compare the total processing time of all jobs with the makespan of S^* and S^{ESL}. A trivial lower bound on the makespan of any feasible schedule—and thus also an optimal schedule—is $\frac{1}{m}\sum_{j\in V} p_j$. The

Earliest Start List Schedule also has to process all the jobs. During a busy period, all m machines process some job, while during an idle period at least one machine processes some job. We therefore get that

$$mT_b + 1T_i \leq \sum_{j \in V} p_j \leq mC_{\max}(S^*).$$

The worst case for $C_{\max}(S^{ESL}) = T_b + T_i$ subject to the presented constraints is achieved for $T_i = C_{\max}(S^*)$ and $T_b = (1 - 1/m)C_{\max}(S^*)$, which yields the result.

Examples for standard precedence graphs that achieve the worst-case bound of Graham's list scheduling can be found in [11] or [7]. If we apply the Earliest Start List Schedule to such instances, there is nothing to be done in the first part of the algorithm and the performance guarantee will be achieved by the last part, the list scheduling. This proves the tightness of the stated approximation ratio and completes the proof. □

We observe that the presence of additional OR constraints does not seem to make it any harder to minimize the makespan of precedence constrained jobs on identical parallel machines. Unfortunately the situation changes completely for the total weighted completion time objective as we will see in the next section.

4 The Total Weighted Completion Time on One Machine

The problem of minimizing the total weighted completion time on one machine is NP-hard in the strong sense as soon as precedence constraints are involved, see [17]. Again this complexity result carries over to AND/OR precedence constraints.

In contrast to the makespan objective, minimizing the total weighted completion time of a set of AND/OR constrained jobs is much harder than for standard precedence constraints. In fact, it is not approximable within any reasonable factor. Therefore we can expect that the successful approaches for standard precedence constraints will not be applicable to AND/OR precedence constraints. We will examine the performance of list scheduling with shortest processing time (SPT) rule on the problem $1|\textit{ao-prec}|\sum \omega_j C_j$ for both cases of unit and arbitrary weights. The SPT rule simply orders the jobs according to non-decreasing processing times. It turns out that the greedy approach of list scheduling with SPT rule results in a useful property of the computed schedule with respect to any other feasible schedule. This property, which is stated in Lemma 4.1, will be a major ingredient for the proofs of the performance guarantees presented thereafter.

Let $(N = (V \cup W, E), p, \omega)$ be an instance of $1|\textit{ao-prec}|\sum \omega_j C_j$ and let S^{SPT} with completion time vector C^{SPT} be the schedule computed for this instance by list scheduling with SPT rule. Consider an arbitrary feasible schedule S and its completion time vector C for (N, p, ω). For any job $j \in V$ we define a threshold $\xi^j(S)$ with respect to schedule S: $\xi^j(S) = \max\{\, p_i \mid C_i \leq C_j \,\}$, that is, the maximum processing time of a job equal to j or scheduled before j in S. $\xi(S) = (\xi^1(S), \ldots, \xi^n(S))$ denotes the vector of thresholds of schedule S.

Lemma 4.1. *Let S be a feasible schedule for an instance (N, p, ω) with threshold $\xi(S)$. Then for every job $j \in V$ it holds that $p_i \leq \xi^j(S)$ for all i with $C_i^{SPT} \leq C_j^{SPT}$.*

Proof. The proof is accomplished by induction along the order of the jobs in the feasible schedule S. To this end we assume without loss of generality that the jobs are numbered such that $S_1 < S_2 < \cdots < S_n$. In addition, we write ξ for short instead of $\xi(S)$.

First we make the following observation for schedule S^{SPT}, which we will refer to as the *greedy choice argument*. Let t be the point in time, at which job j becomes available, that is, in some realization of N all jobs in the predecessor set of j have been completed before t. Then, by the greedy choice of list scheduling with SPT rule, no job with a longer processing time than j itself (and therefore coming after j in the list) will be scheduled between t and j.

Consider job 1. By definition we get that $\xi^1 = p_1$. In addition we know that job 1 is available at time $t = 0$ as it is scheduled at that time in the feasible schedule S and thus cannot have any direct predecessors except s. The claim follows by the greedy choice argument.

Now consider job k and assume that for all $j = 1, \ldots, k-1$ it holds that $p_i \leq \xi^j$ for all i with $C_i^{SPT} \leq C_j^{SPT}$. We have to distinguish between the two cases that the threshold of k is greater or equal to the threshold of $k-1$. Case a) $\xi^k > \xi^{k-1}$. From the definition of ξ it follows that $\xi^k = p_k$ and thus $p_k > \xi^{k-1}$. Let $x \in \{1, \ldots, k-1\}$ be the job that finishes last in the list schedule, that is $C_x^{SPT} = \max\{C_j^{SPT} \mid j = 1, \ldots, k-1\}$. By assumption, $p_i \leq \xi^x \leq \xi^{k-1} < p_k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. We conclude that k is scheduled after x in the list schedule

and $p_i \leq \xi^k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. From the feasibility of schedule S and the maximal choice of x in S^{SPT} it follows that k is available at time C_x^{SPT}. By the greedy choice argument, no job with a strictly longer processing time than k will be scheduled between C_x^{SPT} and C_k^{SPT}, which proves the first case.
Case b) $\xi^k = \xi^{k-1}$. In this case, k may finish before or after some other job $j < k$. Again let $x \in \{1, \ldots, k-1\}$ be such that $C_x^{SPT} = \max\{C_j^{SPT} \mid j = 1, \ldots, k-1\}$. If k completes before x, that is $C_k^{SPT} < C_x^{SPT}$ then the claim trivially holds as by assumption $p_i \leq \xi^x \leq \xi^{k-1} = \xi^k$ for all i with $C_i^{SPT} \leq C_x^{SPT}$. If on the other hand k completes after x, we can again argue as in case a). For the jobs completed before C_x^{SPT} the claim holds by assumption and for the jobs completed between C_x^{SPT} and C_k^{SPT} it follows by the greedy choice argument. □

Note that this property of S^{SPT} holds for the threshold of any feasible schedule S and thus in particular for the threshold of an optimal solution S^*, independent of the optimality criterion.

Theorem 4.2. *Scheduling a set of n AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT) is an $O(\sqrt{n})$-approximation for the problem $1|ao\text{-}prec|\sum C_j$. Moreover, this bound is tight.*

Proof. Consider the schedule S^{SPT} with completion time vector C^{SPT} for an instance (N, p) of $1|ao\text{-}prec|\sum C_j$ with n jobs. Let $x \in V$ be the last job in the list schedule for which x itself and every job completed before x in the list schedule have a processing time smaller than or equal to $C_x^{SPT}/\sqrt{n}$, if such a job exists. Formally, x is chosen such that

$$C_x^{SPT} = \max_{j \in V}\{C_j^{SPT} \mid p_i \leq \frac{C_j^{SPT}}{\sqrt{n}} \ \forall i \text{ with } C_i^{SPT} \leq C_j^{SPT}\}.$$

If such an x exists, let $V^{\leq} = \{j \in V \mid C_j^{SPT} \leq C_x^{SPT}\}$ denote the set of jobs that are scheduled before x including x itself. The set of jobs scheduled after x in the list schedule shall be denoted by $V^{>} = \{j \in V \mid C_j^{SPT} > C_x^{SPT}\}$. If no such x exists, then $V^{\leq} = \emptyset$ and $V^{>} = V$. We have to treat the jobs in the two sets separately.

First consider $V^{\leq}$. The total completion time of the list schedule for the jobs in $V^{\leq}$ can be generously bounded from above by

$$\sum_{j \in V^{\leq}} C_j^{SPT} \leq n C_x^{SPT}.$$

Now we have to bound the total completion time of an optimal schedule for the jobs in $V^{\leq}$ from below. By construction, $p_j \leq C_x^{SPT}/\sqrt{n}$ for every $j \in V^{\leq}$. Thus, there are at least $r \geq \sqrt{n}$ jobs in $V^{\leq}$ and we want to minimize their total completion time. Let $j_1, \ldots, j_r$ be the jobs in $V^{\leq}$ such that $C_{j_1}^* < \cdots < C_{j_r}^*$. Since $\sum_{i=1}^r p_{j_i} \geq C_x^{SPT}$, we obtain the following inequality:

$$\sum_{j \in V^{\leq}} C_j^* \geq C_x^{SPT} + (C_x^{SPT} - p_{j_r}) + (C_x^{SPT} - p_{j_r} - p_{j_{r-1}}) + \cdots + p_1$$

The sum on the right hand side is minimized if we descend from the maximal summand C_x^{SPT} as fast as possible and in as few steps as possible. Since $p_{j_i} \leq C_x^{SPT}/\sqrt{n}$ for $i = 1, \ldots, r$, this is achieved if the processing time of every job j_i, $i = 1, \ldots, r$, takes its maximum possible value $p_{j_i} = C_x^{SPT}/\sqrt{n}$. We then get

$$\sum_{j \in V^{\leq}} C_j^* \geq \sum_{i=1}^{\sqrt{n}} i \, \frac{C_x^{SPT}}{\sqrt{n}} \geq \frac{\sqrt{n}}{2} C_x^{SPT}$$

Combining upper and lower bound yields $\sum_{j \in V^{\leq}} C_j^{SPT} \leq 2\sqrt{n} \sum_{j \in V^{\leq}} C_j^*$ for the jobs in $V^{\leq}$.

Now consider the jobs in $V^{>}$. By definition, for every $j \in V^{>}$ there exists a job k with $p_k > C_j^{SPT}/\sqrt{n}$ that is either equal to j or scheduled before j in the list schedule. Consider an optimal schedule S^* and its corresponding thresholds $\xi^j(S^*)$ for j. By Lemma 4.1 we know that j and every job scheduled before j in the list schedule has a processing time smaller than or equal to $\xi^j(S^*)$. This yields

$$\frac{C_j^{SPT}}{\sqrt{n}} \ < \ p_k \ \leq \ \xi^j(S^*) \ \leq \ C_j^*.$$

Note that $\xi^j(S^*)$ is a trivial lower bound on C_j^* by definition. This is all we need to prove the approximation ratio stated in the theorem, as we can now estimate:

$$\begin{aligned}\sum_{j\in V} C_j^{SPT} \leq & \sum_{j\in V^{\leq}} C_j^{SPT} + \sum_{j\in V^{>}} C_j^{SPT} \\ \leq & \ 2\sqrt{n} \sum_{j\in V^{\leq}} C_j^* + \sum_{j\in V^{>}} \sqrt{n} C_j^* \ \leq \ 2\sqrt{n} \sum_{j\in V} C_j^*\end{aligned}$$

We have shown that list scheduling with SPT rule is a ρ-approximation algorithm with $\rho \in O(\sqrt{n})$, it remains to prove that also $\rho \in \Omega(\sqrt{n})$. Already for standard precedence graphs, the approximation ratio for list scheduling with SPT rule is bounded from below by $\Omega(\sqrt{n})$ and therefore this also holds for the general case of AND/OR precedence constraints. We will demonstrate this by presenting a series of digraphs that force the solution obtained by list scheduling with SPT rule to be a factor of $\Theta(\sqrt{n})$ away from the optimum.

Let $k > 0$ be an integer and consider the following precedence graph $G = (V, E)$. The set of jobs V consists of k jobs $i_1, \dots, i_k$, one job x, and k^2 jobs $j_1, \dots, j_{k^2}$ that form a chain, preceded by x. Thus in E there are the edges (x, j_1) and $(j_\kappa, j_{\kappa+1})$ for all $1 \leq \kappa \leq k^2 - 1$. We have the following job processing times: $p_{i_\kappa} = k^2$, for $\kappa = 1, \dots, k$, $p_x = k^2$, and $p_{j_\kappa} = 1$, for all $\kappa = 1, \dots, k^2$.

The SPT rule orders the jobs according to non-decreasing processing time and thus a possible ordering is $L = j_1, \dots, j_{k^2}, i_1, \dots, i_k, x$. Note that we can force the SPT rule to construct this bad list L by slightly disturbing the processing times. List scheduling then computes a schedule with the following order of the jobs:

$$SPT: \ \boxed{i_1}\boxed{\dots}\boxed{i_k}\boxed{x}\boxed{j_1}\boxed{\dots}\boxed{j_{k^2}}$$

For the objective value of this schedule we get that

$$\sum_{j\in V} C_j^{SPT} \geq \sum_{\ell=1}^{k+1} \ell k^2 + k^2(k+1)k^2 = \Omega(k^5).$$

The optimal schedule prefers job x, which can release the long chain of short jobs. Therefore, an exact algorithm produces the following schedule:

$$OPT: \ \boxed{x}\boxed{j_1}\boxed{\dots}\boxed{j_{k^2}}\boxed{i_1}\boxed{\dots}\boxed{i_k}$$

For the value of an optimal schedule we can calculate that

$$\sum_{j\in V} C_j^* \leq (k^2+1)2k^2 + \sum_{\ell=3}^{k+2} \ell k^2 = O(k^4)$$

Thus the performance ratio of list scheduling for this instance is $\Omega(k^5)/O(k^4) = \Omega(k)$. The precedence graph has $n \in \Theta(k^2)$ many vertices, which yields an approximation guarantee of $\Theta(\sqrt{n})$ for list scheduling with SPT rule on this instance for n respectively k going towards infinity. □

Coming back to the weighted problem $1|\textit{ao-prec}|\sum \omega_j C_j$, we are actually able to establish a strong inapproximability result. Goldwasser and Motwani [9] present an approximation preserving reduction from LABEL COVER to a special case of schedluing with AND/OR precedence constraints arising in disassembly problems.

An instance of a problem considered in [9] is given by a set of jobs, called *tasks*, with unit processing times. The jobs are either AND- or OR-nodes and there are no weights involved. The AND/OR-network representing the precedence constraints is assumed to have *internal-tree* structure, which is a slight generalisation of a tree. We call a node of the network a *leaf*, if it has no direct predecessors. An AND/OR-network N has internal-tree structure, if $N \setminus \{j \in V \mid j \text{ is a leaf}\}$ is an in-tree. The schedluing model requires that for an AND-node to be scheduled, all its direct predecessors have to be processed before the AND-node, and for an OR-node only one direct predecessor has to be scheduled before the OR-node and the others may be left completely unprocessed. The goal of AND/OR schedluing is to minimize a) the number of leaves or b) the number of jobs (AND- and OR-nodes) that need to be scheduled to be able to schedule some sink y of the network, and thus solving the problem instance. Goldwasser and Motwani show that LABEL COVER is a special case of schedluing AND/OR constrained jobs subject to objective a) and that a) can be reduced to b), preserving the approximation guarantee.

Minimizing the number of jobs, i.e. objective b), that need to be scheduled to be able to schedule some sink y of an AND/OR-network can be modeled as a special case of $1|ao\text{-}prec|\sum \omega_j C_j$. In the given AND/OR scheduling instance we assign unit processing times to all AND-nodes and zero processing time to the OR-nodes. In addition we set $\omega_y = 1$ and $\omega_v = 0$ for every other node $v \in V \cup W$. A solution S' to the problem of scheduling a minimum number of AND/OR constrained jobs corresponds to a solution to $1|ao\text{-}prec, p_j = 1|\sum \omega_j C_j$ that minimises the total weighted completion time $\sum_{j \in V} \omega_j C_j = C_y$: Simply schedule all jobs contained in S' before y and all other jobs afterwards. The total weighted completion time is equal to the number of AND-nodes of solution S'.

Together with a strengthened inapproximability result for LABEL COVER by Dinur and Safra [5] we can make the following statement.

Theorem 4.3. *The scheduling problem $1|ao\text{-}prec|\sum \omega_j C_j$ with unit processing times is NP-hard to approximate within a factor of $2^{\log^{1-\gamma} n}$ of the optimum value, where $\gamma = 1/(\log \log n)^c$ for any constant $c < 1/2$ and $n = |V|$ is the number of jobs.*

For a detailed presentation of the proof we refer to [15]. Theorem 4.3 shows that $1|ao\text{-}prec|\sum \omega_j C_j$ is a very hard problem even if all processing times are equal to one. Interestingly enough, the next theorem proves that list scheduling with shortest processing time rule is an easy n-approximation for this problem, and it seems difficult to achieve an asymptotically better ratio.

Theorem 4.4. *Scheduling a set of n weighted AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT) is an n-approximation for the problem $1|ao\text{-}prec|\sum \omega_j C_j$. Moreover, this bound is tight.*

Proof. Let $(N = (V \cup W, E), p, \omega)$ be an instance of $1|ao\text{-}prec|\sum \omega_j C_j$. Without loss of generality assume that the jobs are numbered such that $p_1 \le p_2 \le \cdots \le p_n$. Let S^{SPT} and C^{SPT} denote the schedule and its completion time vector produced by Graham's list schedule with the list $L = 1, \dots, n$. The optimal schedule and its completion time vector are denoted by S^* and C^* as usual. By definition, the threshold $\xi^j(S^*)$ of job j with respect to S^* is a lower bound on C_j^*. Now we have to bound C_j^{SPT} from above. Lemma 4.1 states that for each $j \in V$ it holds that $p_i \le \xi^j(S^*)$ for all jobs i with $C_i^{SPT} \le C_j^{SPT}$. For every $j \in V$ we therefore obtain the following upper bound.

$$C_j^{SPT} \le \sum_{p_i \le \xi^j(S^*)} p_i \le \sum_{p_i \le \xi^j(S^*)} \xi^j(S^*) \le n\xi^j(S^*) \le nC_j^*,$$

which yields the upper bound on the approximation ratio on a per job basis.

To prove the lower bound on the approximation ratio of list scheduling with SPT rule for the problem $1|ao\text{-}prec|\sum \omega_j C_j$, we present a series of AND/OR-networks that force the solution of list scheduling to be a factor of n away from the optimum.

Let $k > 0$ be an integer and consider the following AND/OR-network $N = (V \cup W, E)$. The set of jobs V consists of $k + 2$ jobs, $i_1, \dots, i_k$, x, and j. In addition there is one OR-node $w \in W$. The edge set E contains edges $(i_\kappa, i_{\kappa+1})$ for $\kappa = 1, \dots, k-1$, (i_k, w), (x, w), and (w, j). The processing times of all jobs are equal to one. The weights are as follows, $\omega_{i_\kappa} = 0$ for all $\kappa = 1, \dots, k$, $\omega_x = 0$, and $\omega_j = 1$.

The constructed AND/OR-network is very simple, it consists of one OR-node w with its direct successor j. The two predecessors of the OR-node w are one chain of k nodes and one single node x. If we sort the jobs in order of non-decreasing processing times, we may get the bad list $L = j, i_1, \dots, i_k, x$. Again, by slightly disturbing the processing times we can force the shortest processing time rule to choose this bad list L. List scheduling then computes a schedule with the following order of the jobs:

$$SPT : \begin{array}{|c|c|c|c|c|}\hline i_1 & \dots & i_k & j & x \\ \hline \end{array}$$

For the objective value of this schedule we get that

$$\sum_{j \in V} \omega_j C_j^{SPT} = k + 1.$$

The optimal schedule prefers job x, as it can release j, which is the only job with positive weight. Therefore, an exact algorithm produces the following schedule:

$$OPT : \begin{array}{|c|c|c|c|c|}\hline x & j & i_1 & \dots & i_k \\ \hline \end{array}$$

The value of this optimal schedule is

$$\sum_{j \in V} \omega_j C_j^* = 2.$$

Thus the performance ratio of list scheduling for this instance is $\frac{k+1}{2} = \Theta(k)$. The AND/OR-network has $n = k + 3 \in \Theta(k)$ many vertices, which yields a performance ratio of $\Theta(n)$ for the list scheduling algorithm on this instance for n respectively k going towards infinity. This establishes the lower bound for the approximation guarantee of the shortest processing time rule for the problem $1|ao\text{-}prec|\sum \omega_j C_j$. $\square$

The constructed example has very restricted values, thus the performance guarantee even holds for the problem $1|ao\text{-}prec, p_j = 1, \omega_j \in \{0, 1\}|\sum \omega_j C_j$. In addition, it also provides a lower bound of $\Omega(n)$ for list scheduling with Smith's rule.

We remark that the ideas from the proof of Theorem 4.4 can easily be generalized to show that list scheduling with SPT rule is also an n-approximation algorithm for $P|ao\text{-}prec|\sum \omega_j C_j$.

5 Conclusions

We want to briefly summarize the stated results. In Section 3 we have seen that minimizing the makespan on identical parallel machines subject to AND/OR precedence constraints can be approximated within a factor of 2. This approximation guarantee was established by transforming the AND/OR precedence constraints into standard precedence constraints and applying list scheduling to the resulting instance. It is remarkable that in the case of the makespan objective, additional OR constraints do not seem to make the scheduling problem any harder.

In contrast to this, a major gap occurs in the approximability of minimizing the total weighted completion time for standard and AND/OR precedence constrained jobs on a single machine. While the problem can be approximated within a constant factor for standard constraints, the inapproximability result stated in Theorem 4.3 shows that this is not the case for AND/OR constraints unless P=NP. Therefore, it is not possible to transform the AND/OR constraints into standard precedence constraints such that the approximation ratio for the problem is preserved. In addition, there seems to be a difference in the approximability of the weighted and the unweighted case for AND/OR precedence constraints. For standard precedence constrained jobs, Woeginger proves in [24] that the best possible approximation ratio for minimizing the total weighted completion time is the same for the general problem as well as for a number of special cases including unit processing times, unit weights, and combinations thereof. For AND/OR precedence constrained jobs, the inapproximability result of Theorem 4.3 and the $O(\sqrt{n})$-approximation algorithm for the unweighted case of Theorem 4.2 suggest that a gap may exist between the hardness of approximating the total weighted and the total unweighted completion time. We also could not prove any inapproximability result for $1|ao\text{-}prec|\sum C_j$. It is still possible that there exists a constant or at least logarithmic approximation algorithm for the problem $1|ao\text{-}prec|\sum C_j$.

References

[1] G. M. Adelson-Velsky and E. Levner. Project scheduling in AND/OR graphs: A generalization of Dijkstra's algorithm. Technical report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel, November 1999.

[2] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. R. Rau, and M. Schlansker. Profile-driven instruction level parallel scheduling with application to super blocks. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29)*, pages 58–67, 1996.

[3] C. Chekuri and R. Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98:29–38, 1999.

[4] F. Chudak and D. S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25:199–204, 1999.

[5] I. Dinur and S. Safra. On the hardness of approximating label-cover. Electronic Colloquium on Computational Complexity (ECCC) Technical Report TR99-015, School of Mathematical Sciences, Tel Aviv University, 1999.

[6] M. J. Garey and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. Freemann, New York, 1979.

[7] D. W. Gillies and J. W.-S. Liu. Greed in recource scheduling. *Acta Informatica*, 28:755–775, 1991.

[8] D. W. Gillies and J. W.-S. Liu. Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing*, 24:797–810, 1995.

[9] M. H. Goldwasser and R. Motwani. Intractability of assembly sequencing: Unit disks in the plane. In *Algorithms and Data Structures*, volume 1272 of *LNCS*, pages 307–320. Springer, 1997. Proceedings of the 5th annual Workshop on Algorithms and Data Structures (WADS'97).

[10] M. H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *International Journal of Computational Geometry & Applications*, 9:371–417, 1999.

[11] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[12] 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. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[13] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, pages 513–549, 1997.

[14] J. L. Hennessy and T. Gross. Postpass code optimization of pipeline constraints. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5:4220–448, 1983.

[15] V. Kääb. *Scheduling with AND/OR-Networks*. PhD thesis, Technische Universität Berlin, Germany, 2003.

[16] D. E. Knuth. A generalization of dijkstra's algorithm. *Information Processing Letters*, 6:1–5, 1977.

[17] E. L. Lawler. Sequencing jobs to minimize total weighted completion time. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, Amsterdam, 1993.

[19] F. Margot, M. Queyranne, and Y. Wang. Decompositions, network flows, and a precedence constrained single machine scheduling problem. Technical Report 2000-29, Department of Mathematics, University of Kentucky, Lexington, 2000.

[20] R. H. Möhring, M. Skutella, and F. Stork. Scheduling with AND/OR precedence constraints. Technical Report 689/2000, Technische Universität Berlin, Department of Mathematics, Germany, 2000. To appear in SIAM Journal on Computing.

[21] A. Munier, M. Queyranne, and A. S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *LNCS*, pages 367–383. Springer, 1998. Proceedings of the 6th International Conference on Integer Programming and Combinatorial Optimization (IPCO).

[22] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[23] S. Weiss and J. E. Smith. A study of scalar compilation techniques for pipelined supercomputers. In *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-II)*, pages 105–109, 1987.

[24] G. J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming*, volume 2076 of *LNCS*, pages 887–897. Springer, 2001. Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01).