Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

SCHEDULING SERIES-PARALLEL ORDERS SUBJECT TO 0/1-COMMUNICATION DELAYS

by

ROLF H. MÖHRING   MARKUS W. SCHÄFFTER

No. 611/1998

Scheduling Series-Parallel Orders Subject to 0/1-Communication Delays *

Rolf H. Möhring[†] Markus W. Schäffter[‡]

June 1998

Zusammenfassung

We consider the problem P∞|prec, $c_{ij} \in \{0,1\}|\kappa$ of scheduling jobs with arbitrary processing times on sufficiently many parallel processors subject to series-parallel precedence constraints and 0/1-communication delays in order to minimize a regular performance measure κ. Such schedules without processor restrictions are used for generating approximate solutions for a restricted number of processors.

For unit time communication delays we derive polynomial algorithms to construct optimal schedules when the performance measure κ is the makespan or the average weighted completion time. For n jobs and r precedence constraints, the run times of these algorithms are $\mathcal{O}(n + r)$ and $\mathcal{O}(n^3)$, respectively.

On the other hand, both problems are shown to be NP–hard in the same model for 0/1-communication delays.

Keywords: Scheduling, communication delays, series-parallel order, approximation algorithm

1 Introduction

We consider the problem of scheduling n jobs on sufficiently many (say m) identical parallel processors (as many as jobs suffice, i. e. $m \leq n$) subject to precedence constraints given by a series-parallel order and 0/1-communication delays. Our objective is either the makespan or the average weighted completion time.

This problem has been considered earlier for restricted classes of series-parallel precedence constraints and unit time communication delays.

Chrétienne [Chr89] develops an $\mathcal{O}(n)$ algorithm for minimizing the makespan on in-trees and out-trees.

Chrétienne and Picouleau [CP95] consider a restricted class of series-parallel precedence constraints, in which each block in the decomposition tree (see below) has a unique first and a unique last job that precedes/succeeds every other job in that block. They derive an $\mathcal{O}(q \log q)$ algorithm for the makespan, where $q \leq n$ denotes the number of non-trivial blocks in the decomposition tree.

For the same model, but with unit processing times, Finta et al. [FLMB96] derive an $\mathcal{O}(n^2)$ algorithm for minimizing the makespan on two parallel processors.

We consider precedence constraints that are given by an arbitrary series-parallel order as defined in [VTL82] (the exact definition is given below). For unit time communication delays and arbitrary processing times, we develop a linear-time (in the size of the series-parallel order) algorithm for minimizing the makespan and an $\mathcal{O}(n^3)$ algorithm for minimizing the average weighted completion time. The second objective has not been considered before in this model.

The idea of both algorithms is to use the decomposition tree of the given series-parallel order for constructing an optimal schedule in a bottom up fashion along the tree. The interesting aspect of this approach is that it does not suffice to consider only optimal schedules on subtrees, but that one has to maintain a suitable small set of

*This work was supported by the DFG under grant Mo 446/3-1.

[†]Technische Universität Berlin, Fachbereich Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, e-mail: moehring@math.tu-berlin.de.

[‡]Deutz-Mülheimer Straße 111, 51063 Köln, e-mail: shefta@math.tu-berlin.de.

schedules that may also contain non-optimal schedules with certain "good" properties. For the makespan, this small set consists of at most 2 schedules, while for the average weighted completion time, we need up to $2n$ schedules. This is the main reason for the difference in run time of the two algorithms.

When this model is extended to 0/1-communication delays, we show that the problem of finding a schedule of length at most 6 becomes NP–complete, even for unit processing times. More precisely, we show that the problem P∞|prec=series-parallel, $p_j = 1,\ c_{ij} \in \{0,1\}|C_{\max} < 6$ is NP–complete. The proof consists of a modification of the proof of a similar NP–completeness result by Hoogeveen, Lenstra, and Lenstra [HLV94] for arbitrary orders. As a consequence, this NP–completeness result yields a similar result for the average weighted completion time and a $7/6$ approximation threshold for unit time jobs and both objectives.

The optimal algorithms have an interesting consequence for the construction of approximate solutions for the corresponding scheduling problems with a *restricted* number of processors. A general paradigm to construct such approximate schedules is to take a good (optimal or approximate) schedule for sufficiently many processors, and to exploit information from this schedule (e. g. about actually realized communication delays among jobs). Two (different) such approaches are followed in [HM95, MSS96]. Using an optimal schedule for the makespan constructed by the algorithm presented in this paper with the approach of [MSS96] leads to an approximate schedule for the makespan with a performance guarantee of $(2 - \frac{1}{m})$ when only m processors are available. This is, perhaps surprisingly, the same performance guarantee that is known for the same problem *without* communication delays.

2 Definitions and Notations

Let V with $|V| = n$ denote the set of jobs. Individual jobs are denoted by u, v, w, u_i etc. Precedence constraints are given by a partial order $\Theta = (V, \prec)$ on the set V of jobs, where "$\prec$" is a irreflexive and transitive binary relation (the *ordering relation*) with $u \prec v$ meaning that job u precedes job v.

Let $\Theta_1 = (V_1, \prec_1)$ and $\Theta_2 = (V_2, \prec_2)$ be partial orders with disjoint ground sets V_1, V_2. The *parallel composition* $\Theta_P = (V, \prec_P)$ and the *series composition* $\Theta_S = (V, \prec_S)$ of Θ_1 and Θ_2 are partial orders on $V = V_1 \cup V_2$ whose ordering relations are defined as

$$u \prec_P v :\Leftrightarrow \begin{cases} (u, v \in V_1 \text{ and } u \prec_1 v) \text{ or} \\ (u, v \in V_2 \text{ and } u \prec_2 v) \end{cases}$$

(*parallel composition*)

and

$$u \prec_S v :\Leftrightarrow (u \prec_P v) \text{ or } (u \in V_1 \text{ and } v \in V_2)$$

(*series composition*).

Loosely speaking, in a parallel composition the partial orders Θ_1 and Θ_2 are put "side to side" without any precedences among them, while in a series composition, all of Θ_1 precedes all of Θ_2.

We will use the notation $\Theta_1 \cup \Theta_2$ for the parallel composition and $\Theta_1 \times \Theta_2$ for the series composition. The partial orders Θ_1 and Θ_2 are called the *parallel blocks* and *series blocks* (or simply *blocks*) of Θ_P and Θ_S, respectively. We will often identify these blocks with their ground sets V_1, V_2.

The class of series-parallel orders is then defined recursively as follows: $\Theta = (V, \prec)$ is *series-parallel* if

$|V| = 1$, i. e. Θ is a one-element partial order

or

Θ is obtained by series or parallel composition
of two (smaller) series-parallel orders.

Hence series-parallel orders constitute the smallest class of partial orders that contains the one-element partial order and is closed under parallel and series composition.

As a direct consequence of this recursive definition, every series-parallel order Θ can be represented in a natural way by a (not necessarily unique) binary tree, the *decomposition tree* of Θ. The nodes of the decomposition tree correspond to series-parallel sub-orders. Every inner node is a series or a parallel composition of its children, which are the blocks in the respective composition.

The decomposition tree need not be unique when there are repeated series or parallel compositions, but one can define a unique *canonical decomposition tree* by making repeated compositions of the same kind children of the same node. In this tree, series and parallel compositions alternate along every path in the tree. The canonical decomposition tree thus represents all possible binary decomposition trees. For a given series-parallel order $\Theta = (V, \prec)$ with $n = |V|$ elements and $r = |\prec|$ ordered pairs, a (binary or canonical) decomposition tree can be computed in $\mathcal{O}(n + r)$ time [VTL82].

An example of a series-parallel order and an associated decomposition tree are given below in Figures 2 and 3, respectively. For more details on series-parallel orders, we refer to [Möh89].

The class of scheduling problems with communication delays and sufficiently many identical parallel processors is denoted by $\mathrm{P}\infty|\mathrm{prec}, p_j, c_{ij}|\kappa$. An instance I of this class is given by a tuple $I = (V, \Theta, p, c, \kappa)$, where $V =$ denotes the set of jobs and Θ denotes the (series-parallel in our case) partial order of precedence constraints among the jobs. Each processor can process at most one job at a time, and each job v requires one processor for an uninterrupted period of p_v time units.

If $u \prec v$ and there is no job w with $u \prec w \prec v$, then we denote this by $u \prec\cdot\, v$ and call v a *direct successor* of u and u a *direct predecessor* of v.

For every pair of jobs u, v such that u is a direct predecessor of v in Θ, there is a *communication delay* $c_{uv} \geq 0$. This delay will only occur if job v is scheduled on a different processor than u. It models the time that is required to transfer data from the processor processing u to that processing v. The vector of these communication delays is denoted by c. Notice that the communication delays only depend on the tasks, that any two processors may communicate, and that task replication is not allowed. Throughout the paper, we will only consider the case of 0/1-communication delays, i. e. $c_{uv} \in \{0, 1\}$ for all $u \prec\cdot\, v$.

A *schedule* S for an instance I is a function assigning a *starting time* S_v to each job $v \in V$. Then $C_v = S_v + p_v$ is the *completion time* of job v in S, and C denotes the vector of completion times associated with the schedule. Depending on what is needed, we will use S and C interchangeably to denote a schedule.

Finally, κ is a *regular performance measure* that measures the quality of a schedule. In general, this can be any non-decreasing real-valued function of the completion times C_v, $v \in V$. Here, κ will either be the *makespan* or the *average weighted completion time*. The *makespan* of a schedule S is given by $C_{\max}(S) = \max\{C_v \mid v \in V\}$, while the *total weighted completion time* of S is given by $\sum_{v \in V} w_v\, C_v$, where $w_v \geqslant 0$ is a weight associated with job v. We use $\kappa(S)$ to denote either $C_{\max}(S)$ or $\sum_{v \in v} w_v\, C_v$ and call $\kappa(S)$ the *cost* of schedule S.

A schedule S for an instance I is *feasible* if it satisfies the following constraints:

(1) $S_v \geq \max(\{C_u \mid u \in V,\, u \prec\cdot\, v\})$ for all jobs v,

(2) $S_v < C_u + c_{uv}$ for at most one job $v \in V$ with $u \prec\cdot\, v$ and $c_{uv} > 0$, for all jobs u,

(3) $C_u + c_{uv} > S_v$ for at most one job $u \in V$ with $u \prec\cdot\, v$ and $c_{uv} > 0$, for all jobs v.

Condition (2) ensures that, for every job u, at most one direct successor v is scheduled without communication delay on the same processor as job u. Condition (3) ensures the same for the direct predecessors of a job v. Notice that this definition of feasibility relies on our assumption of 0/1-communication delays and integer processing times p_v.

For a feasible schedule S one can easily construct in linear time a *feasible processor assignment* such that two jobs $u \prec\cdot\, v$ assigned to different processors satisfy $S_v \geqslant C_u + c_{uv}$.

Our algorithms construct a schedule S^∞ along a binary decomposition tree of the given series-parallel order Θ

in a bottom-up fashion. In doing so, we must consider how to obtain the starting times of the jobs in a parallel or series composition from the already computed starting times of the jobs in the corresponding two parallel or series blocks. It turns out that the sub-schedules for the different blocks have to be chosen carefully since a combination of two optimal sub-schedules for blocks B^1 and B^2 need not necessarily give an optimal schedule for the series composition $B = B^1 \times B^2$. This is demonstrated by the following example.

Consider the block B resulting from the parallel composition of the two blocks $B_u = \{u_1, u_2, u_3\}$ and $B_v = \{v_1, v_2, v_3\}$, and suppose that this block B is then used in a series composition with two other blocks B_a and B_c. If B_a and B_c consist of more than one job each, say $B_a = \{a_1, a_2\}$ and $B_c = \{c_1, c_2\}$, then an optimal schedule for $B_a \cup B \cup B_b$ is obtained from an optimal schedule for B as shown in the upper right schedule of Figure 1.

If, however, blocks B_a and B_c consist of one job only, say $B_a = \{a_1\}$ and $B_c = \{c_1\}$, this is no longer true. Instead of using an optimal schedule for B of length 3 with 2 jobs in parallel all the time, we have to choose a sub-optimal schedule of length 4 with only one job at the beginning and the end. The decision which type of schedule will be used later in the algorithm cannot be made locally when block B is considered, but only when the schedule is actually used. So we have to consider and store both possibilities in order to choose the right one at a later step in the computation.

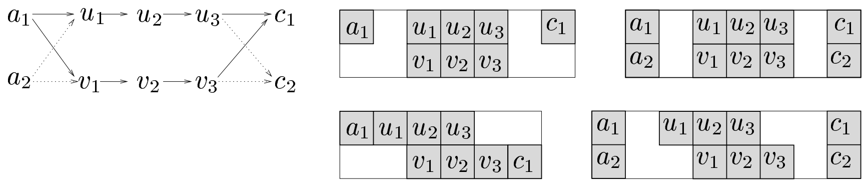

Abbildung 1: A partial order and four possible schedules.

To capture this formally, we introduce the following notation. For a given schedule S^B on a block B, we define its *left interface* left(S^B) as the number of jobs that start at time 0 in S^B. Similarly, we define its *right interface* right(S^B) as the number of jobs completed at time $C_{\max}(S^B)$. It makes of course only sense to consider schedules S with left$(S) > 0$ and right$(S) > 0$.

Interfaces model the different cases how schedules of two blocks can be put together in a parallel or series composition. The crucial case is always given when one has to compose schedules S^1, S^2 of two series blocks B^1, B^2, and there are two or more jobs in the last time slot of S^1 or in the first time slot of S^2. In this case, one must insert an extra time slot for the communication delays. The construction along the decomposition tree is such that this bad situation is avoided as much as possible.

We call the pair $(\max\{2, \text{left}(S^B)\}, \max\{2, \text{right}(S^B)\})$ the *interface type* of S^B.

The problem of finding an optimal schedule for a series-parallel order Θ on B is now reduced to the problem of finding an optimal composition of *feasible*, maybe *sub-optimal* schedules for the blocks of Θ. The crucial question then is, how many schedules we have to consider on a block.

For the minimum makespan, we will show that it suffices to store only two feasible schedules for each block. This is done in the next section. For the average weighted completion time, we have to store up to $2|B| - 1$ sub-schedules per block B, but still arrive at a polynomial algorithm. This is done in Section 4.

An important notion for the following analysis is that of *strongly optimal schedules*. A schedule S on a block B of the decomposition tree is called *strongly optimal* with respect to performance measure κ if

(i) S is of minimum cost $\kappa(S)$ among all feasible schedules on B,

(ii) $C_{\max}(S)$ is minimum among all optimal schedules on B,

(iii) right(S) is minimum among all schedules satisfying (i) and (ii).

Note that for the minimum makespan objective, Conditions (i) and (ii) coincide.

3 Minimizing the Makespan

The algorithm follows a given binary series-parallel decomposition tree in a bottom-up fashion starting with all singletons, i. e. blocks that only consist of one job. For each block B, the algorithm maintains a strongly optimal schedule S^B as well as a schedule $\bar{S}^B$ that is strongly optimal among all feasible schedules S on B with left$(S) = 1$.

For two schedules S' and S'' on two blocks B^1 and B^2, let $S' \cup S''$ denote the straight-forward parallel composition of the schedules S' and S'', i. e. the starting time $(S' \cup S'')_v$ of job v in schedule $S' \cup S''$ equals S'_v if $v \in B^1$ and S''_v if $v \in B^2$. Clearly, $S' \cup S''$ is a feasible schedule on $B = B^1 \cup B^2$.

Similarly, we define the schedule $S' \times S''$ for a series composition $B_1 \times B_2$ as follows. $(S' \times S'')_v$ is set to S'_v if $v \in B^1$ and $C_{\max}(S') + S''_v$ if $v \in B^2$. Notice that $S' \times S''$ does not necessarily respect the communication delays between jobs of B^1 and jobs of B^2. For a schedule S, let $S + 1$ denote the schedule that is obtained by increasing the starting time of each job v in S by one time unit: $(S + 1)_v = S_v + 1$.

With this notation, we can describe the algorithm as follows. For each block B, select from the possible schedules $(S^1, \bar{S}^1)$ and $(S^2, \bar{S}^2)$ stored for the children blocks B^1 and B^2 of B those combinations of schedules that lead to a strongly optimal schedule S^B and a strongly optimal schedule $\bar{S}^B$ with left interface 1 for block B, respectively.

In the case of a parallel composition $B = B^1 \cup B^2$, $S^1 \cup S^2$ is the best combination for S^B, while for $\bar{S}^B$, one of $\bar{S}^1 \cup (S^2 + 1)$ and $\bar{S}^2 \cup (S^1 + 1)$ is the best choice.

For a series composition $B = B^1 \times B^2$, the situation is much simpler. If right$(S^1) \geqslant 2$, an additional time slot must be introduced between the schedules on B^1 and B^2. Hence, $S^B = S^1 \times (S^2 + 1)$ is strongly optimal in this case. For the other case that right$(S^1) = 1$, the schedule $S^B = S^1 \times \bar{S}^2$ is a best choice (as is $S^1 \times (S^2 + 1)$ if S^2 is shorter than $\bar{S}^2$). For $\bar{S}^B$, observe that the restriction of $\bar{S}^B$ to B^1 is strongly optimal on B^1. Hence it suffices to consider $\bar{S}^1$ instead of S^1 and apply the same arguments as above.

As an example of this computation, consider the series-parallel order in Figure 2. Here, each job is represented by a box whose length represents the job's processing time. A decomposition tree of this series-parallel order is given in Figure 3 together with, for each block B, both schedules, S^B and $\bar{S}^B$. If only one schedule is given, then $S^B = \bar{S}^B$.

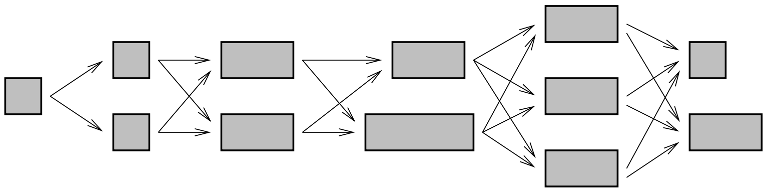

Abbildung 2: An example series-parallel order.

The correctness of the algorithm is proven by the following lemma.

Lemma 3.1. *Consider an instance $I = (m, V, p, \Theta)$ of the problem P∞|prec=series-parallel,$c_{ij} = 1|C_{\max}$, and let B be a block a binary decomposition tree of Θ. Let S^B and $\bar{S}^B$ denote the schedules constructed by the above algorithm for block B. Then:*

(i) *S^B is a strongly optimal schedule on B.*

(ii) *left$(\bar{S}^B) = 1$ and $\bar{S}^B$ is strongly optimal among all feasible schedules S' on B with left$(S') = 1$.*

Beweis. The proof is done by induction along the decomposition tree. Obviously Lemma 3.1 holds for singleton blocks.

So consider a block B and assume that the lemma has already been proven for all blocks below B. Let $(S^1, \bar{S}^1)$ and $(S^2, \bar{S}^2)$ denote the schedules constructed by the algorithm for the children blocks B^1 and B^2 of B in the

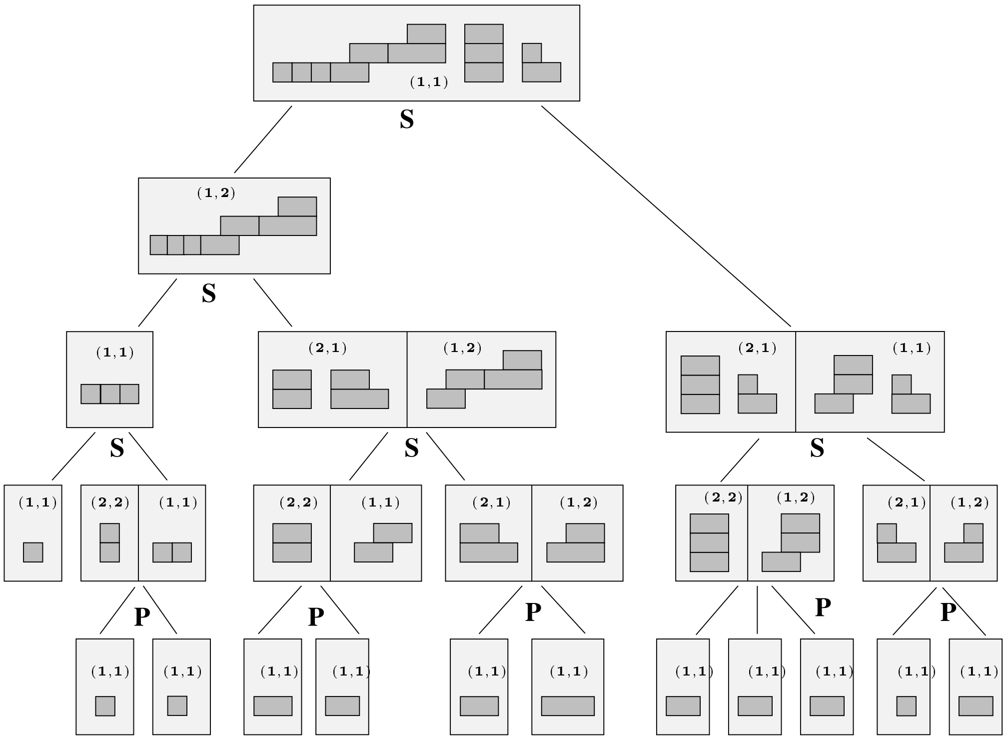

Abbildung 3: The composition of sub–schedules.

decomposition tree.

Consider a strongly optimal schedule $S^\star$ on B. Notice that $S^\star$ decomposes into two disjoint sub-schedules on B^1 and B^2, respectively. In the case of a series composition, this is obvious since every job of B^2 is a successor of every job of B^1. For a parallel composition, it can be assumed without loss of generality that the jobs of B^1 and B^2 are scheduled on different sets of processors since there are sufficiently many processors available and there is no communication delay between a job from B^1 and a job from B^2.

An easy case analysis then yields that these sub-schedules correspond exactly to one of the combinations of a strongly optimal and/or a strongly optimal schedule with left interface 1 on the blocks B_1 and B_2. By the inductive hypothesis, these schedules are available on the blocks B_1 and B_2. Hence the best combination among them leads to a strongly optimal schedule on B. □

Lemma 3.1 can be used to compute the optimal makespan and also an optimal schedule along the decomposition tree. The run-time of this algorithm is as follows.

Theorem 3.2. *For $P\infty|prec$ = series-parallel, $c_{ij} = 1|C_{\max}$, the above algorithm can be implemented to construct an optimal schedule in $\mathcal{O}(n+r)$ time, where n and r denote the number of jobs and the number of precedence constraints of Θ, respectively.*

Beweis. The decomposition tree can be constructed in $\mathcal{O}(n + r)$ time [VTL82]. The straightforward construction of the schedules takes $\mathcal{O}(|B|)$ time per node B of the tree, since the starting times of the jobs in B need to be updated. Since the tree has at most $2n - 1$ nodes, the total time spent on the construction of schedules along the tree would be $\mathcal{O}(n^2)$.

However, one can do better by a *delayed* calculation of the starting times combined with storing only $\mathcal{O}(1)$ information in every tree node B. The main idea behind this is the fact that the update of the starting time of all jobs v in B from the same sub-block B^i consists in adding the same *offset* to all jobs $v \in B_i$ (e. g. the length of S^1, or just 1 etc.), depending on the case analysis in the proof of Lemma 3.1.

Hence the information needed in a tree node B can be reduced to the length of the two schedules S^B and $\bar{S}^B$, their interface types, and their offsets. This information can obviously obtained from that of the two children in $\mathcal{O}(1)$ time, as the case analysis in the proof of Lemma 3.1 shows.

The actual starting times of the jobs can then be determined by an additional top-down computation in the decomposition tree, in which the offsets of a node are added to the offset of the corresponding child. This takes $\mathcal{O}(1)$ time per tree node. Finally, the thus accumulated offsets of a leaf $B = \{v\}$ determine the starting time S_v of job v.

Since the tree has at most $2n - 1$ nodes, this takes $\mathcal{O}(n)$ time in total. □

In the case of forests and single source/single target series-parallel orders considered by [CP95] (i. e., a series-parallel order where each block contains exactly one minimal and one maximal job), the occurring interface types are much simpler. In the case of (in-)forests, all right interfaces equal 1 and hence, also 0/1-communication delays can be dealt with (see [MS95]). For single source/single target series-parallel orders, the differentiation between several interface types is no longer necessary at all, since for each block, a feasible schedule contains exactly one job in its first and its last time slot, respectively.

Our approach can be extended to 0/1-communication delays if all communication delays are locally identical, i. e. one of the following conditions holds:

- $c_{u,v} = c_{u,w}$ for all jobs $u, v, w \in V$ with $u \prec\!\cdot v, u \prec\!\cdot w \in \Theta$,
- $c_{u,v} = c_{w,v}$ for all jobs $u, v, w \in V$ with $u \prec\!\cdot v, w \prec\!\cdot v \in \Theta$,

or if Θ is a single source/single target series-parallel order in the sense of [CP95].

In contrast, the problem with $0/1$-communication delays is NP-hard for arbitrary series-parallel orders.

Theorem 3.3. *The problem P∞|prec=series-parallel, $p_j = 1$, $c_{ij} \in \{0, 1\}|C_{\max} < 6$ is NP–complete.*

Beweis. The proof consists of a modification of a proof by Hoogeveen, Lenstra and Veltman [HLV94] who showed that the problem P∞|prec, $p_j = 1$, $c_{ij} = 1|C_{\max} < 6$ is NP–complete. The main idea is to insert precedence constraints with 0-communication delays in order to make the partial order used in [HLV94] series-parallel, while preserving all other properties of their transformation.

The transformation is from the NP–complete problem 3–SATISFIABILITY (3–SAT). An instance of 3–SAT consists of a finite set U of variables, a collection C of clauses c over U, each clause consisting of exactly 3 literals. The question is whether there exists a variable assignment that satisfies each clause $c \in C$ (called a TRUE–*assignment*).

The details of this transformation are as follows. For each variable and each clause we introduce a set of jobs and precedence pairs as depicted in Figure 4.

For every variable v_i, we introduce the jobs v_i^0, v_i^1, v_i^2, v_i^3, v_i, $\bar{v}_i$, and v_i^5 with the precedence constraints among them as depicted in the upper part of Figure 4.

The construction for jobs that correspond to a clause c depends on the occurrences of the variables in c. Consider a clause $c \in C$ and denote the three variables that occur negated or unnegated in c by v_1, v_2, v_3. Let x_i denote the literal corresponding to v_i, i. e. either $x_i = v_i$ or $x_i = \bar{v}_i$. For each variable v_i we introduce three jobs: $\hat{v}_i^c$, v_i^c, $\bar{v}_i^c$. We add a job for each possible pair of literals occurring in c: (x_1, x_2), (x_1, x_3), (x_2, x_3). Each such job (x_i, x_j) is the successor of the two variable jobs that correspond to the literals x_i and x_j: if $x_i = v_i$ then $v_i^c \prec (x_i, x_j)$ and $\bar{v}_i^c \prec (x_i, x_j)$ otherwise. The same is done for x_j.

To complete the transformation given by Hoogeveen, Lenstra, and Veltman include a precedence constraint between job v_i^c and v_i and between $\bar{v}_i^c$ and $\bar{v}_i$ for each variable $v_i \in U$ and each clause that contains v_i or $\bar{v}_i$.

Note that the partial order defined so far is *not* series-parallel. We therefore (and this is the modification of the transformation of Hoogeveen, Lenstra, and Veltman) introduce additional precedence constraint as follows. Assign a communication delay of 1 to each precedence constraint introduced so far and introduce a precedence constraint with a communication delay of 0 between every maximal job in U_1 and every minimal job in U_2 (if they are still unrelated) as depicted in Figure 4.

This ensures that U_1 and U_2 are series blocks in a series composition. It is easy to see from Figure 4 that U_1 and U_2 are themselves series-parallel. Hence the whole modified partial order is series-parallel.

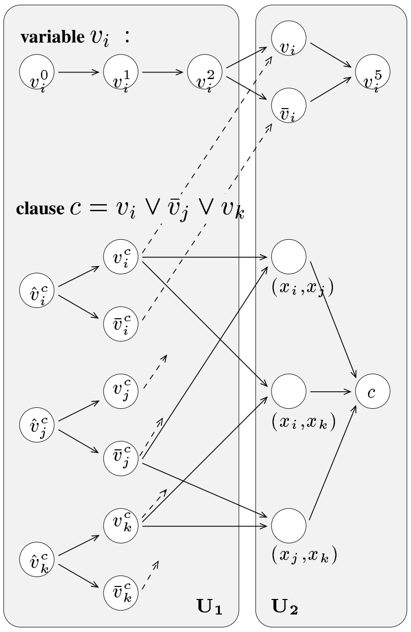

Abbildung 4: Transformation from 3–SAT.

We will now show that there is a feasible schedule of length 6 for the constructed scheduling instance if and only if there is a TRUE–assignment for the given 3–SAT instance. We start by making two essential observations. Due to the precedence constraints, a schedule of length 6 is requires that all jobs in U_1 are completed before time 3. Note that no job of U_2 can be scheduled before time 3.

Consider now a variable $v_i \in U$. Due to the unit time communication delays between the jobs v_i and $\bar{v}_i$ and their successor v_i^5, in every feasible schedule of length 6 one of the jobs v_i and $\bar{v}_i$ is scheduled at time 3 while the other is scheduled at time 4.

Similarly, the unit time communication delays between $\hat{v_i}^c$ and its direct successors v_i^c and $\bar{v}_i^c$ ensure that in every feasible schedule of length 6 one job of $\{v_i^c, \bar{v}_i^c\}$ is scheduled at time 1 while the other is scheduled at time 2.

Assume now that there exists a TRUE–assignment for the given instance of the 3–SAT problem. Then a schedule of length 6 is obtained by starting job v_i at time 3 and job $\bar{v}_i$ at time 4 if the corresponding variable v_i is assigned the value TRUE, and vice-versa otherwise. The remaining jobs corresponding to variable v_i are scheduled in a canonical way at times 0, 1, 2 and 5 and hence are completed before time 6.

Analogously, start v_i^c at time 1 and $\bar{v}_i^c$ at time 2 if variable v_i occurs unnegated in clause c and vice-versa otherwise. The remaining jobs are scheduled in a straight-forward way. Since at least one literal in each clause is TRUE, at least two of the three jobs (x_i, x_j) of each clause can be scheduled at time 3 and hence, all jobs corresponding to some clauses are completed before time 6.

In the converse direction we must show that every feasible schedule S of length 6 yields a TRUE–assignment for the given instance of the 3–SAT problem. Define variable v_i to be TRUE if job v_i is scheduled at time 3 by S and to be FALSE otherwise. Consider a clause $c \in C$. Since job c is completed before time 6, only one job of the jobs (x_a, x_b), say (x_i, x_j), can be started after time 4. The remaining jobs $(x_i, x_k),\ (x_j, x_k)$ then have to be started at time 3. This yields that for at least one variable v_i, the job of $\{v_i^c, \bar{v}_i^c\}$ that corresponds to the literal occurring in c has to be completed at time 2. Without loss of generality assume that this is v_i^c. Then job $\bar{v}_i^c$ must be scheduled after time 2 due to the communication delays between $\hat{v}_i^c$ and its direct successors. Notice that job $\bar{v}_i$ cannot be scheduled before time 4 since two of its predecessors (v_i^2 and $\bar{v}_i^c$) are not completed before time 3. Hence, $\bar{v}_i$ is started at time 4 in S and due to the communication delays between V_i^5 and its direct predecessors, job v_i has to be started at time 3 which means that variable v_i is assigned to TRUE and hence satisfies clause c.

Since this holds for every clause c, the variable-assignment satisfies every clause. This completes the proof of Theorem 3.3. □

As a direct consequence, we obtain:

Corollary 3.4. *Unless P = NP, there is no polynomial time approximation algorithm for P*∞*|prec=series-parallel,* $p_j = 1,\ c_{ij} \in \{0,1\}|C_{\max}$ *with a performance ratio strictly better than* $7/6$.

4 Minimizing Average Weighted Completion Time

The approach of the previous section can be extended to the average weighted completion time as performance measure. However, the number of non-optimal schedules that we have to store in each block of the decomposition tree increases. As a simple example for this increase consider a sequence of series blocks B_i, each consisting of a parallel composition of a job a_i of length 3 (or a chain of 3 unit time jobs) and a unit time job b_i, together with a single job c succeeding every other job.

A schedule with minimum makespan for this example is given in the left part of Figure 5. For regular performance measures other than the makespan, it may be necessary to schedule job b_i before job a_i for some blocks B_i in order to achieve optimality. Note that even for the average weighted completion time objective, i. e. $\kappa = \sum w_j C_j$, every combination depicted in Figure 5 can become optimal depending on the weights of the different jobs.

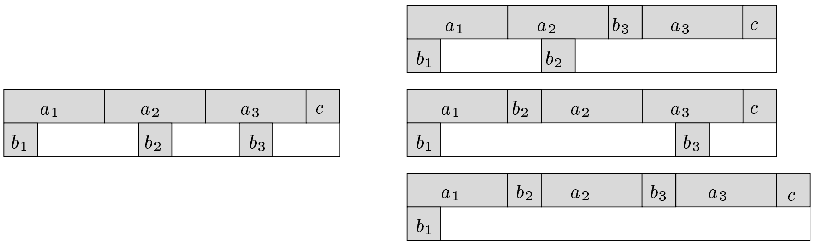

Abbildung 5: Possible optimal schedules for $\kappa = \sum w_j C_j$.

Hence for the average weighted completion time, we have to store more sub-optimal schedules for a block B than for the makespan. Let $\kappa_{\min}(B,\ell)$ denote the minimum cost of a schedule for the jobs of block B of length at most ℓ. Similarly, let $\bar{\kappa}_{\min}(B,\ell)$ denote the minimum cost among all schedules of length at most ℓ with a left interface of 1. (Of course, we consider only schedule lengths ℓ for which a feasible schedule of length at most ℓ exists.)

In order to reduce the number of considered schedules, we store schedules only for those lengths ℓ that are not dominated by shorter schedules, i. e. for which $\kappa_{\min}(B,l) < \kappa_{\min}(B,l-1)$ or $\bar{\kappa}_{\min}(B,l) < \bar{\kappa}_{\min}(B,l-1)$.

Note that, for the makespan, this domination rule just leads to the 2 schedules considered in the previous section. Thus the approach followed here for $\kappa = \sum w_j C_j$ can be seen as a natural extension of the approach for the makespan.

We will show that all these schedules can be constructed efficiently in a bottom up fashion along a binary decomposition tree, and that the number of different schedule lengths ℓ that we must consider is bounded by the number of jobs in the current block.

This eventually gives a minimum cost schedule for each schedule length for the root block of the decomposition tree and hence for the considered instance of the problem P∞|prec=series-parallel, $p_j = 1,\ c_{ij} = 1|\kappa = \sum w_j C_j$.

We now describe the algorithm in detail. For each singleton block, i. e. a block that consists of only one job v, only one schedule is stored: job v starts at time 0 and the schedule length is $\ell = p_v$.

Consider now a non-singleton block B. In the binary decomposition tree, B is composed of exactly two sub-blocks B^1 and B^2. The algorithm now computes all possible compositions of schedules S^1, S^2 stored for B^1 and B^2, respectively. In the case of a series composition $B = B^1 \times B^2$, only combinations of the kind $S^1 \times S^2$ (remember the notations introduced in Section 3) are considered, while in the case of a parallel composition, all combinations of the kind $S^1 \cup S^2$, $S^1 \cup (S^2 + 1)$ and $(S^1 + 1) \cup S^2$ are investigated.

For each schedule length ℓ occurring in these combinations, determine a strongly optimal schedule S_ℓ^{B} (among all considered combinations) and a schedule $\bar{S}_\ell^{\mathrm{B}}$ that is strongly optimal among all combined schedules S of length at most ℓ and with left$(S) = 1$. Store schedules S_ℓ^{B} only for those schedule lengths ℓ that satisfy $\kappa_{\min}(B, \ell) < \kappa_{\min}(B, \ell - 1)$ and schedules $\bar{S}_\ell^{\mathrm{B}}$ only if $\bar{\kappa}_{\min}(B, \ell) < \bar{\kappa}_{\min}(B, \ell - 1)$.

Lemma 4.1. *Consider an instance $I = (m, V, p, \Theta, \kappa)$ of the problem P|prec=series-parallel, $c_{ij} = 1| \sum w_j C_j$, and let B be a block of a binary decomposition tree of Θ. Let $\ell_1 < \ell_2 < \ldots < \ell_k$ denote the different lengths for which schedules are stored for B by the algorithm. Let L be any positive integer and let $\ell = argmax\{\ell_i \leq L \mid i = 1, \ldots, k\}$. Then:*

- **(i)** *$\kappa_{\min}(B, \ell_1) > \ldots > \kappa_{\min}(B, \ell_k)$, $\bar{\kappa}_{\min}(B, \ell_1) > \ldots > \bar{\kappa}_{\min}(B, \ell_k)$ and $\ell_k - \ell_1 \leq |B| - 1$,*
- **(ii)** *S_ℓ^B is strongly optimal among all feasible schedules on B of length at most L,*
- **(iii)** *$\bar{S}_\ell^B$ is strongly optimal among all feasible schedules S on B of length at most L with left$(S) = 1$.*

Beweis. The proof is similar to the proof of Lemma 3.1. Obviously Lemma 4.1 holds for singleton blocks. Hence consider a block B composed of blocks B^1 and B^2 and assume that the lemma has already been shown for B^1 and B^2.

Statement (i) follows from the construction of the algorithm (only schedule lengths are considered that are not dominated by smaller lengths) and the following observation. Let $\ell_{\min}^{(i)}$ and $\ell_{\max}^{(i)}$ denote the minimum and maximum length for which a schedule is stored for block B^i, $i = 1, 2$. If $B = B^1 \times B^2$, then

$$\ell_k - \ell_1 \leq \ell_{\max}^{(1)} - \ell_{\min}^{(1)} + 1 + \ell_{\max}^{(2)} - \ell_{\min}^{(2)} \leq |B^1| - 1 + 1 + |B^2| - 1 = |B| - 1.$$

Note that there is at most one idle time slot between the two partial schedules on B^1 and B^2 since we consider unit time communication delays.

On the other hand, if $B = B^1 \cup B^2$, then

$$\begin{aligned} \ell_k - \ell_1 &\leq \max(\ell_{\max}^{(1)}, \ell_{\max}^{(2)}) - \max(\ell_{\min}^{(1)}, \ell_{\min}^{(2)}) \\ &\leq \max(\ell_{\max}^{(1)} - \ell_{\min}^{(1)}, \ell_{\max}^{(2)} - \ell_{\min}^{(2)}) \\ &\leq \max(|B^1|, |B^2|) - 1 < |B| - 1. \end{aligned}$$

This proves Statement (i).

The following argumentation works for both cases in a similar way. For (ii) consider a schedule $S^\star$ on B that is optimal among all feasible schedules of length at most L while, for (iii), let $S^\star$ denote a schedule on B that is optimal among all feasible schedules S of length at most L and with left$(S) = 1$.

In the following, we only consider Statement (ii). Statement (iii) follows analogously.

Notice that $S^\star$ decomposes into two disjoint sub-schedules $S_1^\star$, $S_2^\star$ on B^1 and B^2 as in the proof of Lemma 3.1. Let $\ell_1^\star$ and $\ell_2^\star$ be the lengths of these two schedules, respectively.

By the inductive hypothesis there exist two schedules S^1, S^2 stored for B^1 and B^2 of length at most $\ell_1^\star$ and $\ell_2^\star$, respectively. Both schedules are strongly optimal on B^1 and B^2 among all schedules of length at most $\ell_1^\star$ and $\ell_2^\star$, respectively.

Hence for a parallel composition $B = B^1 \cup B^2$,

$$\kappa(S^\star) = \kappa(S_1^\star) + \kappa(S_2^\star) \overset{\text{(hyp.)}}{\geq} \kappa(S^1) + \kappa(S^2) = \kappa(S^1 \cup S^2)\,.$$

Note that $S^1 \cup S^2$ is one of the compositions computed by the algorithm. Hence, there exists a schedule S stored for B that satisfies

- $\kappa(S) \leq \kappa(S^\star)$,
- $\text{right}(S) \leq \text{right}(S^\star)$,
- $C_{\max}(S) = \max(C_{\max}(S^1),\, C_{\max}(S^2)) \leq \max(\ell_1^\star,\, \ell_2^\star) = C_{\max}(S^\star) \leq L.$

Now it follows directly from the assumptions on $S^\star$ that in the case of a parallel composition, S is strongly optimal among all schedules on B of length at most L.

For a series composition $B = B^1 \times B^2$, we distinguish two cases.

Case 1: $\text{right}(S_1^\star) \geq 2$ or $\text{left}(S_2^\star) \geq 2$.

In this case, there is an idle time slot between the sub-schedules $S_1^\star$ and $S_2^\star$ in $S^\star$. Hence,

$$\kappa(S^\star) = \kappa(S_1^\star) + \kappa(S_2^\star) + w(B^2) \cdot (\ell_1^\star + 1)$$

with $w(B^2) = \sum_{v \in B^2} w_v$. Note that this is the place in the proof where we need the fact that $\kappa = \sum w_j C_j$. Since $C_{\max}(S^1) \leq \ell_1^\star$ and $C_{\max}(S^1) \leq \ell_1^\star$, it follows that $\kappa(S^1) \leq \kappa(S_1^\star)$ and $\kappa(S^2) \leq \kappa(S_2^\star)$. The sub-schedules S^1 and S^2 can trivially be composed by inserting an empty time slot between them. We thus obtain

$$\begin{aligned} \kappa(S^\star) &\geq \kappa(S^1) + \kappa(S^2) + w(B^2) \cdot (C_{\max}(S^1) + 1) \\ &= \kappa(S^1 \times S^2)\,. \end{aligned}$$

Case 2: $\text{right}(S_1^\star) = \text{left}(S_2^\star) = 1$.

We can assume without loss of generality that $S_1^\star$ is strongly optimal among all schedules on B^1 of length at most $\ell_1^\star$ and with a left interface of 1. By the inductive hypothesis, there is a schedule S^1 stored for B^1 with $\kappa(S_1^\star) = \kappa(S^1)$, $C_{\max}(S^1) = \ell_1^\star$ and $\text{right}(S^1) = 1$.

Similarly, we have stored a schedule S^2 for B_2 that is strongly optimal among all feasible schedules with a left interface of 1 and length at most $\ell_2^\star$. Hence $\kappa(S^2) \leq \kappa(S_2^\star)$.

Putting this together leads to

$$\begin{aligned} \kappa(S^\star) &= \kappa(S_1^\star) + \kappa(S_2^\star) + w(B^2) \cdot \ell_1^\star \\ &\geq \kappa(S^1) + \kappa(S^2) + w(B^2) \cdot C_{\max}(S^1) \\ &= \kappa(S^1 \times S^2)\,. \end{aligned}$$

In both cases, $S^1 \times S^2$ is one of the compositions computed by the algorithm and hence, there exists a length $\ell_i \leq L$ for which a schedule S^{B} is stored which is strongly optimal among all schedules on B of length at most L. This completes the proof of Lemma 4.1. □

Because of Statement (i) of Theorem 4.2, the number of different schedules stored for each block B is bounded by $2(|B|-1)$. Since the algorithm traverses a binary decomposition tree and since for each block B, the number of combinations looked at by the algorithm is bounded by $2(|B^1|-1) \cdot 2(|B^2|-1) \in \mathcal{O}(|V|^2)$, and since updating the information for the at most $2(|B|-1)$ different schedules of B takes $2|B|(|B|-1)$ time, the overall computing time of the algorithm is at most $\mathcal{O}(|V|^3)$. Note that the idea of delayed decisions by using offsets as in Theorem 3.2 does not lead to a better run-time since one needs to inspect a quadratic number of combinations to fine the best schedules for a node B. This yields the following theorem.

Theorem 4.2. *For $P\infty|prec$ = series-parallel, $c_{ij} = 1|\sum w_j C_j$, the above algorithms constructs an optimal schedule for every upper bound of the schedule length in $\mathcal{O}(|V|^3)$ time.*

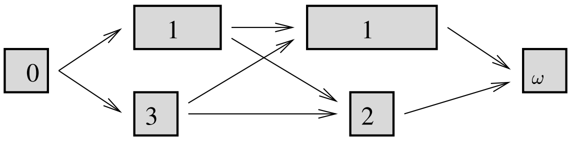

Abbildung 6: An example for $\kappa = \sum w_j C_j$.

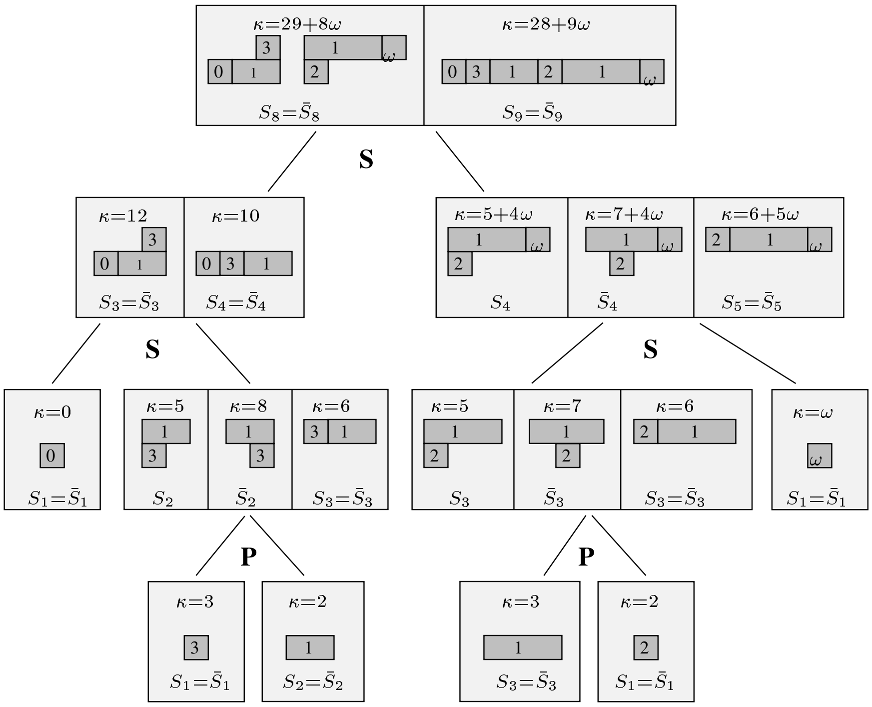

Abbildung 7: The composition of sub–schedules.

For a demonstration of the algorithm, consider the series-parallel order in Figure 6. Each job is represented by a box whose length corresponds to the processing time of the job. The number inside each box denotes the weight of the corresponding job.

The decomposition tree in Figure 7 displays the different schedules stored for each block. Note that we have omitted the trivial singleton blocks in the middle of the tree.

The weight ω of the last job of Θ is left undefined in order to demonstrate which schedules will actually be stored by the algorithm. For $\omega \geq 1$, the schedule S_8 of the root block is optimal for all schedule lengths (8 is the minimum makespan). On the other hand, for $\omega = 0$ schedule S_8 remains the only optimal schedule of length at most 8 but schedule S_9 becomes optimal for all schedule lengths $\ell \geq 9$.

Similar to Theorem 3.3 we obtain that the problem with $0/1$-communication delays is NP-hard.

Theorem 4.3. *The problem P∞|prec=series-parallel, $p_j = 1$, $c_{ij} \in \{0, 1\}| \sum w_j C_j < 6$ is NP–complete.*

Beweis. The proof is based on a straightforward reduction from P∞|prec=series-parallel, $p_j = 1$, $c_{ij} \in \{0, 1\}|C_{\max} < 6$. Consider an instance I of this problem. Insert a dummy job z that succeeds every other job of instance I. Clearly, the resulting partial order of precedence constraints is again series-parallel. Now give job z the weight $w_z = 1$ and all other jobs v the weight $w_v = 0$. This defines an instance I' of P∞|prec=series-parallel, $p_j = 1$, $c_{ij} \in \{0, 1\}| \sum w_j C_j < 6$. Obviously, $\sum w_j C_j < 6$ for I' iff $C_{\max} < 6$ for I. □

Again, we obtain as a consequence:

Corollary 4.4. *Unless P = NP, there is no polynomial time approximation algorithm for P∞|prec=series-parallel, $p_j = 1$, $c_{ij} \in \{0, 1\}| \sum w_j C_j$ with a performance ratio strictly better than $7/6$.*

For arbitrary partial orders, it has recently been shown by Hoogeveen, Schuurman, and Woeginger [HSW98] that the problem P∞|prec, $p_j = 1$, $c_{ij} = 1| \sum C_j$ (i. e., with identical weigths $w_{ij} = 1$) is NP-complete and cannot be approximated with a performance ratio strictly better than $9/8$ (unless P = NP).

5 Concluding Remarks

The above approach does not hold for arbitrary regular performance measures. Figure 8 gives an counter example for $\kappa(S) = \max(2 + S(y), 2S(z))$ on a set of three jobs, $V = \{x, y, z\}$ of length $p_x = 2$, $p_y = p_z = 1$.

Consider first the set $\{y, z\}$. An optimal schedule with a left interface of 1 is given by scheduling job y before job z (denote this schedule by S_{yz}). In contrast, for $V = \{x\} \times \{y, z\}$, the only optimal schedule contains the sub-optimal schedule S_{zy} on $\{y, z\}$ that schedules z before y.

The reason for this is that the schedule $S_{yz} + 1$ is not optimal while S_{yz} is. Hence, optimality depends in a complicated way on the number of time units by which a schedule is shifted in the composition of sub-schedules. For $\kappa = \sum w_j C_j$ this can be handled. For general κ, however, we do not see a way.

In fact, the proof of Lemma 4.1 uses the properties of the average weighted completion time that an optimality is preserved under shifting and that the resulting cost is the original cost plus a value that only depends on the amount of time by which the schedule is shifted but not on the particular schedule itself. Lemma 4.1 remains valid for all cost functions κ with this property.

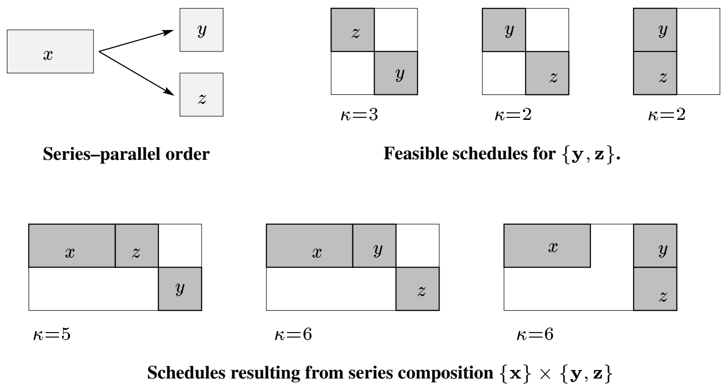

Abbildung 8: A counter-example for arbitrary regular cost functions.

Acknowledgment

We thank two referees for their detailed comments on an earlier version of the paper and their valuable hints on the literature.

Literatur

[Chr89] P. Chrétienne, *A polynomial time algorithm to optimally schedule tasks over an ideal distributed system under tree-like precedence constraints*, European J. Oper. Res. **43** (1989), 225–230.

[CP95] P. Chrétienne and C. Picouleau, *Scheduling with communication delays: A survey*, Scheduling Theory and its Applications (P. Chretienne, E. G. Coffman, and J. K. Lenstra, eds.), John Wiley & Sons, New York, 1995, pp. 65–90.

[FLMB96] L. Finta, Z. Liu, I. Milis, and E. Bampis, *Scheduling UET–UCT series-parallel graphs on two processors*, Theor. Comp. Sc. **162** (1996), no. 2, 323–340.

[HLV94] J. A. Hoogeveen, J. K. Lenstra, and B. Veltman, *Three, four, five, six, or the complexity of scheduling with communication delays*, Operations Research Letters **16** (1994), 129–137.

[HM95] C. Hanen and A. Munier, *An approximation algorithm for scheduling dependent tasks on m processors with small communication delays*, IEEE Symposium on Emerging Technologies and Factory Automation, september 1995.

[HSW98] H. Hoogeveen, P. Schuurman, and G. J. Woeginger, *Non-approximability results for scheduling problems with minsum criteria*, Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, 1998, To appear.

[Möh89] R. H. Möhring, *Computationally tractable classes of ordered sets*, Algorithms and Order (I. Rival, ed.), Nato Advanced Study Institutes Series, D. Reidel Publishing Company, Dordrecht, 1989, pp. 105–193.

[MS95] R. H. Möhring and M. W. Schäffter, *A simple approximation algorithm for scheduling forests with unit processing times and zero-one communication delays*, Preprint No. 506/1995, Department of Mathematics, University of Technology, Berlin, Germany, 1995.

[MSS96] R. H. Möhring, M. Schäffter, and A. S. Schulz, *Scheduling jobs with communication delays: using infeasible solutions for approximation*, Algorithms — ESA'96, Fourth Annual European Symposium, Springer-Verlag, Lecture Notes in Computer Science, vol. 1136, 1996, pp. 76–90.

[VTL82] J. Valdes, R. E. Tarjan, and E. L. Lawler, *The recognition of series parallel digraphs*, SIAM Journal on Computing **11** (1982), no. 2, 298–313.

Reports from the group

"Combinatorial Optimization and Graph Algorithms"

of the Department of Mathematics, TU Berlin

689/2000 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Scheduling with AND/OR precedence constraints

682/2000 *Rolf H. Möhring and Marc Uetz:* Scheduling Scarce Resources in Chemical Engineering

681/2000 *Rolf H. Möhring:* Scheduling under Uncertainty: Optimizing Against a Randomizing Adversary

680/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Solving Project Scheduling Problems by Minimum Cut Computations (Journal version merging the two Reports 620 and 661)

674/2000 *Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia:* Optimal Covering Tours with Turn Costs

669/2000 *Michael Naatz:* A Note On a Question of C. D. Savage

667/2000 *Sándor P. Fekete and Henk Meijer:* On Geometric Maximum Weight Cliques

666/2000 *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the Continuous Weber and k-Median Problems

664/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* A Note on Scheduling Problems with Irregular Starting Time Costs

661/2000 *Frederik Stork and Marc Uetz:* Resource-Constrained Project Scheduling: From a Lagrangian Relaxation to Competitive Solutions

658/1999 *Olaf Jahn, Rolf H. Möhring, and Andreas S. Schulz:* Optimal Routing of Traffic Flows with Length Restrictions in Networks with Congestion

655/1999 *Michel X. Goemans and Martin Skutella:* Cooperative facility location games

654/1999 *Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang:* Single Machine Scheduling with Release Dates

653/1999 *Andreas S. Schulz and Martin Skutella:* Scheduling unrelated machines by randomized rounding

646/1999 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Forcing Relations for AND/OR Precedence Constraints

640/1999 *Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko:* Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates

639/1999 *Andreas S. Schulz and Martin Skutella:* The Power of α-Points in Preemptive Single Machine Scheduling

634/1999 *Karsten Weihe, Ulrik Brandes, Annegret Liebers, Matthias Müller–Hannemann, Dorothea Wagner and Thomas Willhalm:* Empirical Design of Geometric Algorithms

633/1999 *Matthias Müller–Hannemann and Karsten Weihe:* On the Discrete Core of Quadrilateral Mesh Refinement

632/1999 *Matthias Müller–Hannemann:* Shelling Hexahedral Complexes for Mesh Generation in CAD

631/1999 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing Weighted b-Matching Algorithms: Insights from a Computational Study

629/1999 *Martin Skutella:* Convex Quadratic Programming Relaxations for Network Scheduling Problems

628/1999 *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

627/1998 *Jens Gustedt:* Specifying Characteristics of Digital Filters with FilterPro

620/1998 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource Constrained Project Scheduling: Computing Lower Bounds by Solving Minimum Cut Problems

619/1998 *Rolf H. Möhring, Martin Oellrich, and Andreas S. Schulz:* Efficient Algorithms for the Minimum-Cost Embedding of Reliable Virtual Private Networks into Telecommunication Networks

618/1998 *Friedrich Eisenbrand and Andreas S. Schulz:* Bounds on the Chvátal Rank of Polytopes in the 0/1-Cube

617/1998 *Andreas S. Schulz and Robert Weismantel:* An Oracle-Polynomial Time Augmentation Algorithm for Integer Proramming

616/1998 *Alexander Bockmayr, Friedrich Eisenbrand, Mark Hartmann, and Andreas S. Schulz:* On the Chvátal Rank of Polytopes in the 0/1 Cube

615/1998 *Ekkehard Köhler and Matthias Kriesell:* Edge-Dominating Trails in AT-free Graphs

613/1998 *Frederik Stork:* A branch and bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints

612/1998 *Rolf H. Möhring and Frederik Stork:* Linear preselective policies for stochastic project scheduling

611/1998 *Rolf H. Möhring and Markus W. Schäffter:* Scheduling series-parallel orders subject to 0/1-communication delays

609/1998 *Arfst Ludwig, Rolf H. Möhring, and Frederik Stork:* A computational study on bounding the makespan distribution in stochastic project networks

605/1998 *Friedrich Eisenbrand:* A Note on the Membership Problem for the Elementary Closure of a Polyhedron

596/1998 *Andreas Fest, Rolf H. Möhring, Frederik Stork, and Marc Uetz:* Resource Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates

595/1998 *Rolf H. Möhring Andreas S. Schulz, and Marc Uetz:* Approximation in Stochastic Scheduling: The Power of LP-based Priority Policies

591/1998 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing Weighted b-Matching Algorithms: Towards a Flexible Software Design

590/1998 *Stefan Felsner and Jens Gustedt and Michel Morvan:* Interval Reductions and Extensions of Orders: Bijections to Chains in Lattices

584/1998 *Alix Munier, Maurice Queyranne, and Andreas S. Schulz:* Approximation Bounds for a General Class of Precedence Constrained Parallel Machine Scheduling Problems

577/1998 *Martin Skutella:* Semidefinite Relaxations for Parallel Machine Scheduling

Reports may be requested from: Hannelore Vogt-Möller
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: moeller@math.TU-Berlin.DE

Reports are also available in various formats from

`http://www.math.tu-berlin.de/coga/publications/techreports/`

and via anonymous ftp as

`ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-`*number–year*`.ps`